

For Review Purposes Only

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



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

Part No: 820-4803-10
October 31, 2008

Copyright 2008 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more U.S. patents or pending patent applications in the U.S. and in other countries.

U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, the Solaris logo, the Java Coffee Cup logo, docs.sun.com, Java, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and SunTM Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Products covered by and information contained in this publication are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical or biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2008 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuelle relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plusieurs brevets américains ou des applications de brevet en attente aux Etats-Unis et dans d'autres pays.

Cette distribution peut comprendre des composants développés par des tierces personnes.

Certains composants de ce produit peuvent être dérivés du logiciel Berkeley BSD, licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays; elle est licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, le logo Solaris, le logo Java Coffee Cup, docs.sun.com, Java et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui, en outre, se conforment aux licences écrites de Sun.

Les produits qui font l'objet de cette publication et les informations qu'il contient sont régis par la législation américaine en matière de contrôle des exportations et peuvent être soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes chimiques ou biologiques ou pour le nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers des pays sous embargo des Etats-Unis, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régis par la législation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFACON.

List of Remarks

REMARK 2-3	Reviewer	I'm unclear about the reach of the <code>ssoadm</code> command-line utility. Can it be used to edit properties in the bootstrap file? 36
REMARK 2-4	Reviewer	I suppose that I should mention policy response attributes here, no? 37
REMARK 2-5	Reviewer	A customer once complained about the roles used in this section. He thought it was confusing to use the role "Employee" Anyway, see below that I've changed both "Employee" and "Buyer" to "Bidder." Is this a good change to make? 40
REMARK 3-3	Reviewer	I am not confident about the explanation above or the example below. Is this the correct property? Please make suggestions as appropriate. 54
REMARK 3-5	Reviewer	Above, is it accurate to mention the bootstrap file and only the bootstrap file? 55
REMARK 3-6	Reviewer	I'm confused here. Is the property correct in the line of code below? 56
REMARK 3-7	Reviewer	For the line above, what about this property?: <code>am.encrypted.pwd</code> . Does it apply instead? 56
REMARK 3-8	Reviewer	Is the above description accurate? 64
REMARK 4-2	Reviewer	Above, I say one can set properties in the bootstrap file using the <code>ssoadm</code> utility. Is that true? 67
REMARK 4-7	Reviewer	What's with this <code>com.ipplanet.am.naming.url</code> ? Does it not exist in 3.0. Should another property be used instead or should this write up be completely different? 82
REMARK A-1	Reviewer	Is this write up accurate? Does everything in this wildcard section fit the agent type? ..97
REMARK C-1	Reviewer	Please look at the command above. Are all the details of it accurate. It seems that the above example is different for web agents and J2EE agents. Besides that example above, does this appendix apply equally to web agents and J2EE agents? Are there any differences that should be addressed? 115

Contents

Preface	9
1 Introduction to Policy Agent 3.0	15
Overview of New Features in Policy Agent 3.0	15
Compatibility and Coexistence of Policy Agent 3.0 with Previous Releases	17
Compatibility of Policy Agent 3.0 with Access Manager 7.1 and Access Manager 7 2005Q4	17
Coexistence of Policy Agent 3.0 With Policy Agent 2.2	17
2 Role of Policy Agent 3.0 Software	19
An Overview of Policy Agent 3.0	19
A Generalized Example of the Policy Decision Process	23
Examples of the Policy Decision Process by Agent Type	28
Policy Agent 3.0 and the Centralized Agent Configuration Option	33
Web Agents and J2EE Agents: Similarities and Differences	34
J2EE Agents in the Policy Agent 3.0 Release	39
Uses of J2EE Agents	39
J2EE Agents and an Online Auction Application	40
J2EE Agents and a Web-Based Commerce Application	40
J2EE Agents and a Content-Based Web Application	41
How J2EE Agents Work	41
Using the J2EE Agent Sample Application in Policy Agent 3.0	42
The Sample Application	42
3 Vital Installation Information for a J2EE Agent in Policy Agent 3.0	43
Format of the Distribution Files for a J2EE Agent Installation in Policy Agent 3.0	44
▼ To Unpack Non-Package Formatted Deliverables of a J2EE Agent in Policy Agent 3.0	45

▼ To Unpack Package Formatted Deliverables of a J2EE Agent in Policy Agent 3.0	45
▼ To Unpack a .zip Compressed file of a J2EE Agent in Policy Agent 3.0	46
Role of the agentadmin Program in a J2EE Agent for Policy Agent 3.0	47
agentadmin --install	48
agentadmin --uninstall	49
agentadmin --listAgents	51
agentadmin --agentInfo	52
agentadmin --version	52
agentadmin --encrypt	53
agentadmin --getEncryptKey	55
agentadmin --uninstallAll	56
agentadmin --getUuid	57
agentadmin --usage	58
agentadmin --help	59
J2EE Agent Directory Structure in Policy Agent 3.0	59
Location of the J2EE Agent Base Directory in Policy Agent 3.0	60
Inside the J2EE Agent Base Directory in Policy Agent 3.0	60
Configuring A J2EE Agent With Access Manager 7.0	64
Creating a J2EE Agent Profile	64
▼ To Create an Agent Profile	65
 4 Common J2EE Agent Tasks and Features in Policy Agent 3.0	67
Common J2EE Agent Tasks and Features	67
Hot-Swap Mechanism in J2EE Agents	68
J2EE Agent Properties That Are List Constructs	69
J2EE Agent Properties That Are Map Constructs	71
J2EE Property Configuration: Application Specific or Global	72
J2EE Agent Filter Modes	74
Enabling Web-Tier Declarative Security in J2EE Agents	76
Enabling Failover in J2EE Agents	81
Login Attempt Limit in J2EE Agents	83
Redirect Attempt Limit in J2EE Agents	83
Not-Enforced URI List in J2EE Agents	84
Fetching Attributes in J2EE Agents	85
Configuring FQDN Handling in J2EE Agents	90

Using Cookie Reset Functionality in J2EE Agents	91
Enabling Port Check Functionality in J2EE Agents	92
Key Features and Tasks Performed With the J2EE agentadmin Program	93
Key Features and Tasks Performed With the J2EE Agent API	94
Class AmFilterManager	94
Interface IAmSSOCache	95
Class AmSSOCache	95
Usage of New J2EE Agent API in Policy Agent 3.0	96
A Wildcard Matching in Policy Agent 3.0 J2EE Agents	97
The Multi—Level Wildcard: *	98
The One-Level Wildcard: -*	99
B Precautions Against Session-Cookie Hijacking in an OpenSSO EnterpriseDeployment	101
Defining Key Cookie Hijacking Security Issues	101
OpenSSO Enterprise Solution: Cookie Hijacking Security Issues	105
OpenSSO Enterprise Solution: Shared Session Cookies	105
OpenSSO Enterprise Solution: A Less Secure Application	106
OpenSSO Enterprise Solution: Modification of Profile Attributes	106
Key Aspects of the OpenSSO Enterprise Solution: Cookie Hijacking Security Issues	106
Implementing the OpenSSO Enterprise Solution for Cookie Hijacking Security Issues	108
Creating or Updating an Agent Profile	108
Configuring the OpenSSO Enterprise Deployment Against Cookie Hijacking	111
C Using the ssoadm Command-Line Utility With Agents	115
An ssoadm Command-Line Example Specific to Agents	115
Listing the Options for an ssoadm Subcommand	116
Agent-Related Subcomands for the ssoadm Command	118
The ssoadm Command: add-agent-to-grp subcommand	118
The ssoadm Command: agent-remove-props subcommand	119
The ssoadm Command: create-agent subcommand	120
The ssoadm Command: create-agent-grp subcommand	121
The ssoadm Command: delete-agent-grps subcommand	122
The ssoadm Command: delete-agents subcommand	123

The ssoadm Command: list-agent-grp-members subcommand	124
The ssoadm Command: list-agent-grps subcommand	125
The ssoadm Command: list-agents subcommand	125
The ssoadm Command: remove-agent-from-grp subcommand	126
The ssoadm Command: show-agent subcommand	127
The ssoadm Command: show-agent-grp subcommand	128
The ssoadm Command: show-agent-membership subcommand	129
The ssoadm Command: show-agent-types subcommand	130
The ssoadm Command: update-agent subcommand	130
The ssoadm Command: update-agent-grp subcommand	131
 Index	 133

Preface

The Sun OpenSSO Enterprise Policy Agent software consists of J2EE (Java 2 Platform Enterprise Edition) agents and web agents. This *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for J2EE Agents* provides an overview of how J2EE agents work in the Sun OpenSSO Enterprise Policy Agent 3.0 release, while also providing a description of the features and processes of Policy Agent that are the same for all J2EE agents and web agents. The J2EE agents and web agents have many similarities, but the two types of agents also have some differences. This guide starts with a summary of the similarities and differences. Then this guide follows with information only applicable to J2EE agents. However, for information for specific J2EE agents, such as installation information and agent-specific configuration, see the individual J2EE agent guide for that agent.

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

Who Should Use This Book

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

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

Before You Read This Book

You should be familiar with documentation related to 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, documents in the OpenSSO Enterprise 8.0 documentation set are available at the following location:

Be aware that in the future, newer versions of OpenSSO Enterprise might also apply to Policy Agent 3.0. In such cases, see the appropriate OpenSSO Enterprise documentation set.

- OpenSSO Enterprise 8.0 Installation Instructions
- OpenSSO Enterprise 8.0 Core Documentation

Updates to the *Release Notes* and links to modifications of the core documentation can be found in the OpenSSO Enterprise documentation collection.

Policy Agent 3.0 Documentation Set

This *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for J2EE Agents* is available in two documentation sets: the OpenSSO Enterprise documentation set and in the Policy Agent 3.0 documentation set as described in this section. The other guides in the Policy Agent 3.0 documentation set are described in the following sections:

- “[Individual Agent Guides](#)” on page 10 (each agent has its own guide)
- “[Release Notes](#)” on page 11

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

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

The individual agent guides are listed along with supported server information in the Policy Agent 3.0 Release Notes.

Release Notes

The Policy Agent 3.0 Release Notes are released online after an agent or set of agents is released. The release notes include a description of what is new in the current release, known problems and limitations, installation notes, and how to report issues with the software or the documentation.

Related Sun Microsystems Product Documentation

The information in the table that follows specifies the key document collections that might be involved in anOpenSSO Enterprise deployment, which are available at the following location:

<http://docs.sun.com/prod/entsys.05q4>

TABLE P-1 Documentation Collections Related to

Title	Location
Directory Server:	http://docs.sun.com/coll/1316.1
Web Server:	http://docs.sun.com/coll/1308.1
Application Server:	http://docs.sun.com/coll/1310.1
Message Queue:	http://docs.sun.com/coll/1307.1
Web Proxy Server:	http://docs.sun.com/coll/1311.1

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.

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 *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for J2EE Agents*, and the part number is 820-4803.

Documentation, Support, and Training

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

Typographic Conventions

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

TABLE P-2 Typographic Conventions

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

Shell Prompts in Command Examples

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

TABLE P-3 Shell Prompts

Shell	Prompt
C shell prompt	<code>machine_name%</code>
C shell superuser prompt	<code>machine_name#</code>
Bourne shell and Korn shell prompt	<code>\$</code>
Bourne shell and Korn shell superuser prompt	<code>#</code>

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 the `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 agent administrator no longer needs to edit an agent's `AMAgent.properties` file.

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 server where the agent is installed. For example, the agent profile name and password used to access 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.

- More hot-swappable agent configuration properties

Version 3.0 agents 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 in URL policy

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 A, “Wildcard Matching in Policy Agent 3.0 J2EE Agents”](#)

- Default J2EE 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.
- **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.

- Option to create the agent profile for J2EE agents 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.

- Option to lock the agent configuration properties

Changing configuration properties can have unexpected results. Furthermore, hot-swappable properties take effect immediately. Therefore, configuration mistakes are instantly implemented. In the Policy Agent 3.0 release you have a method for locking the configuration to help prevent such accidental changes.

The Policy Agent 3.0 properties

- Automated migration support

You can migrate a version 2.2 agent to a version 3.0 agent using the `agentadmin` program with the `-migrate` option.

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

Compatibility and Coexistence of Policy Agent 3.0 with Previous Releases

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

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

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

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

The `com.sun.identity.agents.config.repository.location` property in the OpenSSO Enterprise server Agent Service schema (`AgentService.xml` file) specifies where the agent configuration data is stored:

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

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. However, agents in the 2.2 release must continue to store their configuration data locally in the `AMAgent.properties` file. And because the configuration data of agents in the 2.2 release is local to the agent, OpenSSO Enterprise centralized agent configuration is not supported. To configure an agent in the Policy Agent 2.2 release, you must continue to edit the `AMAgent.properties` file.

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

Role of Policy Agent 3.0 Software

OpenSSO EnterprisePolicy Agent 3.0 software consists of web agents and J2EE agents. This chapter explains the similarities and differences of these two types of agents. Then the chapter describes J2EE agents in more detail.

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.

All agents do the following:

- Enable cross-domain single sign-on (CDSSO).
- 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.

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 as depicted by the following figures.

The figure that follows is J2EE-agent specific.

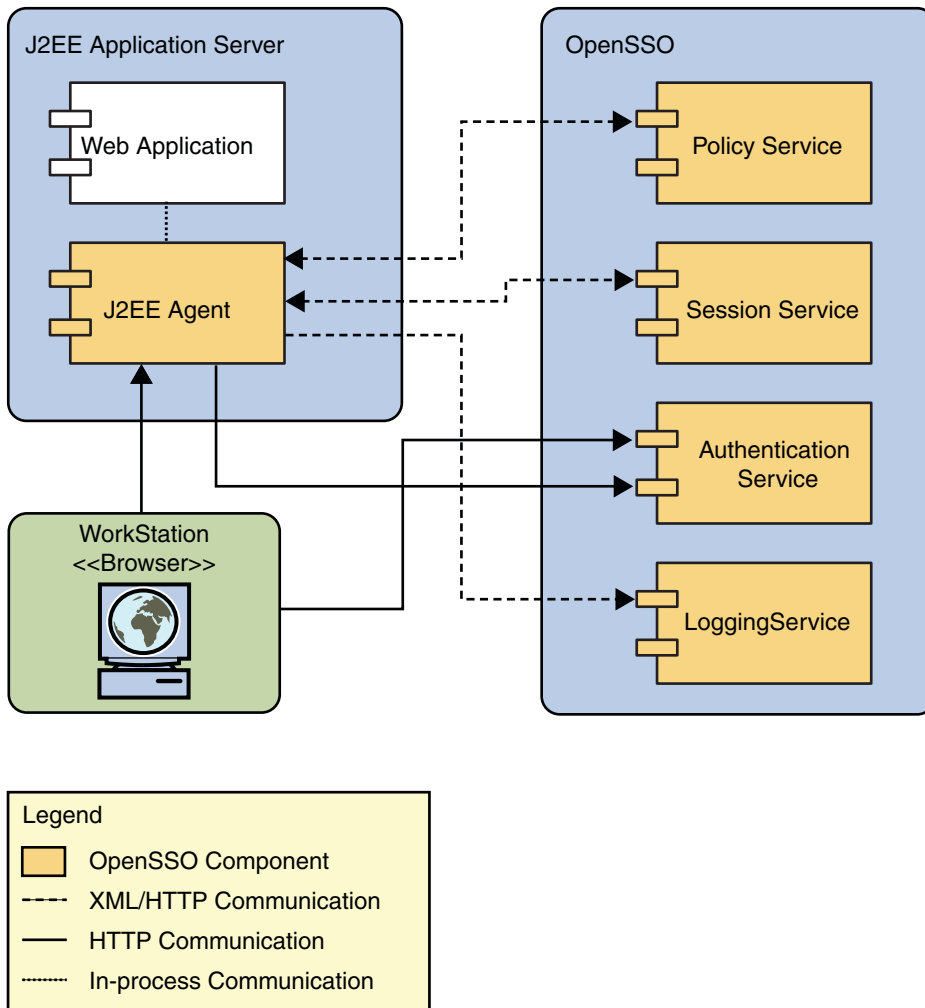


FIGURE 2-1 J2EE Agent Interaction with OpenSSO Enterprise Services

Notice the slight differences in how each agent type (J2EE agents in the preceding figure and web agents in the following figure) interacts with the OpenSSO Enterprise Services. Therefore, the two agent types tend to achieve the same results, but sometimes with slightly different processes.

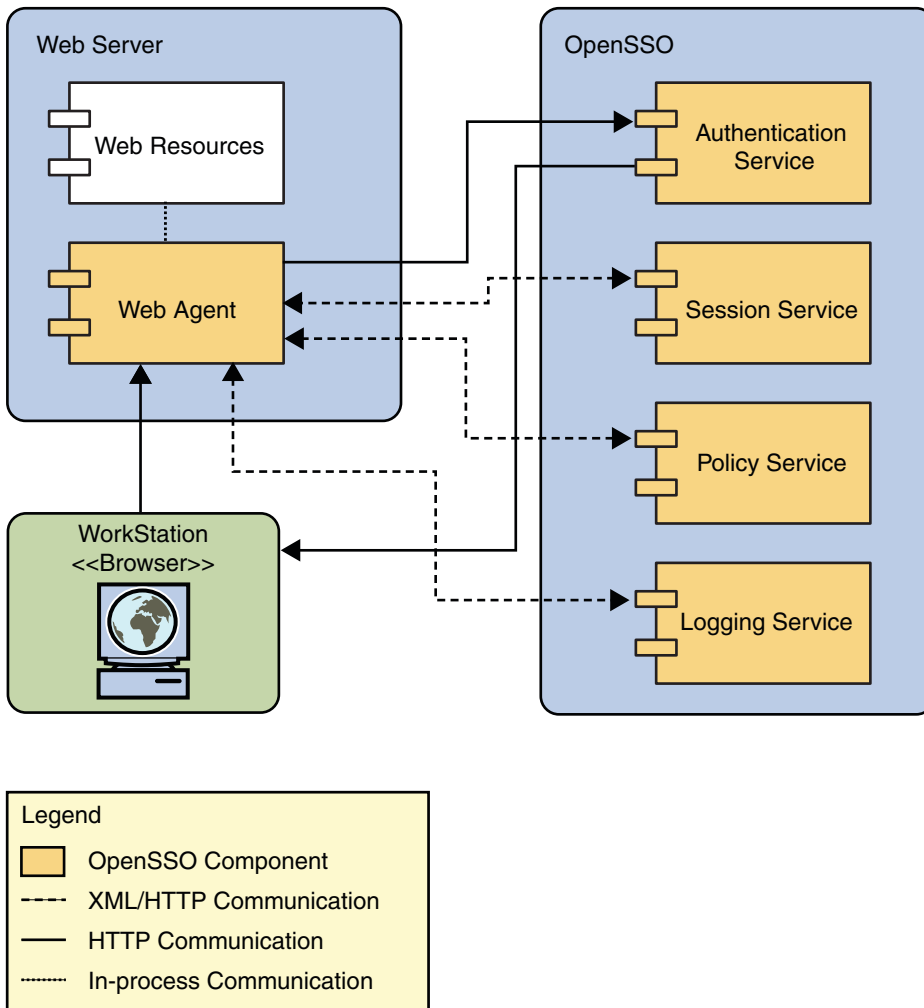


FIGURE 2-2 Web Agent Interaction with OpenSSO Enterprise Services

A key point is that the agent must interact continuously with various OpenSSO Enterprise services. For both J2EE agents and web agents, the interaction starts between the agent and the OpenSSO Enterprise Authentication Service. After authentication, users still cannot access a protected resource until the defined policies regarding user privileges are approved. The agent and OpenSSO Enterprise continue to interact, performing several small tasks back and forth, until the agent finally enforces a policy decision to either allow or deny access. 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) would seem to handle these deployments in the same manner. For example, observe how the two figures that follow are nearly identical. In these two figures J2EE agents and web agents incorporate firewalls and load balancers in the same manner.

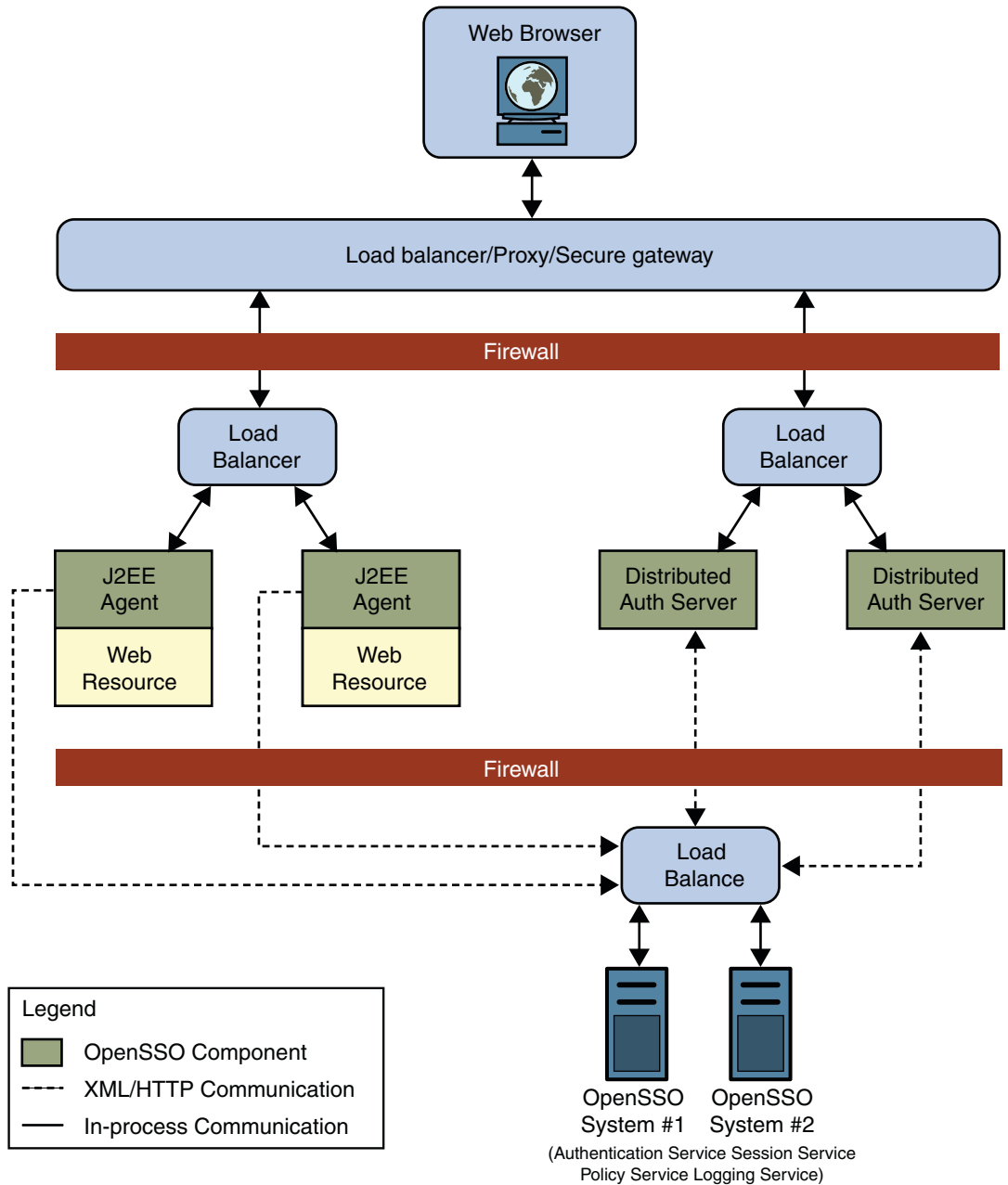


FIGURE 2-3 High Level View of an OpenSSO Enterprise Deployment With J2EE Agents, Firewalls, and Load Balancers

By comparing the preceding figure with the figure that follows, you can see that from this level of detail J2EE agents and web agents do not differ in the general role they play in an OpenSSO Enterprise deployment. 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.

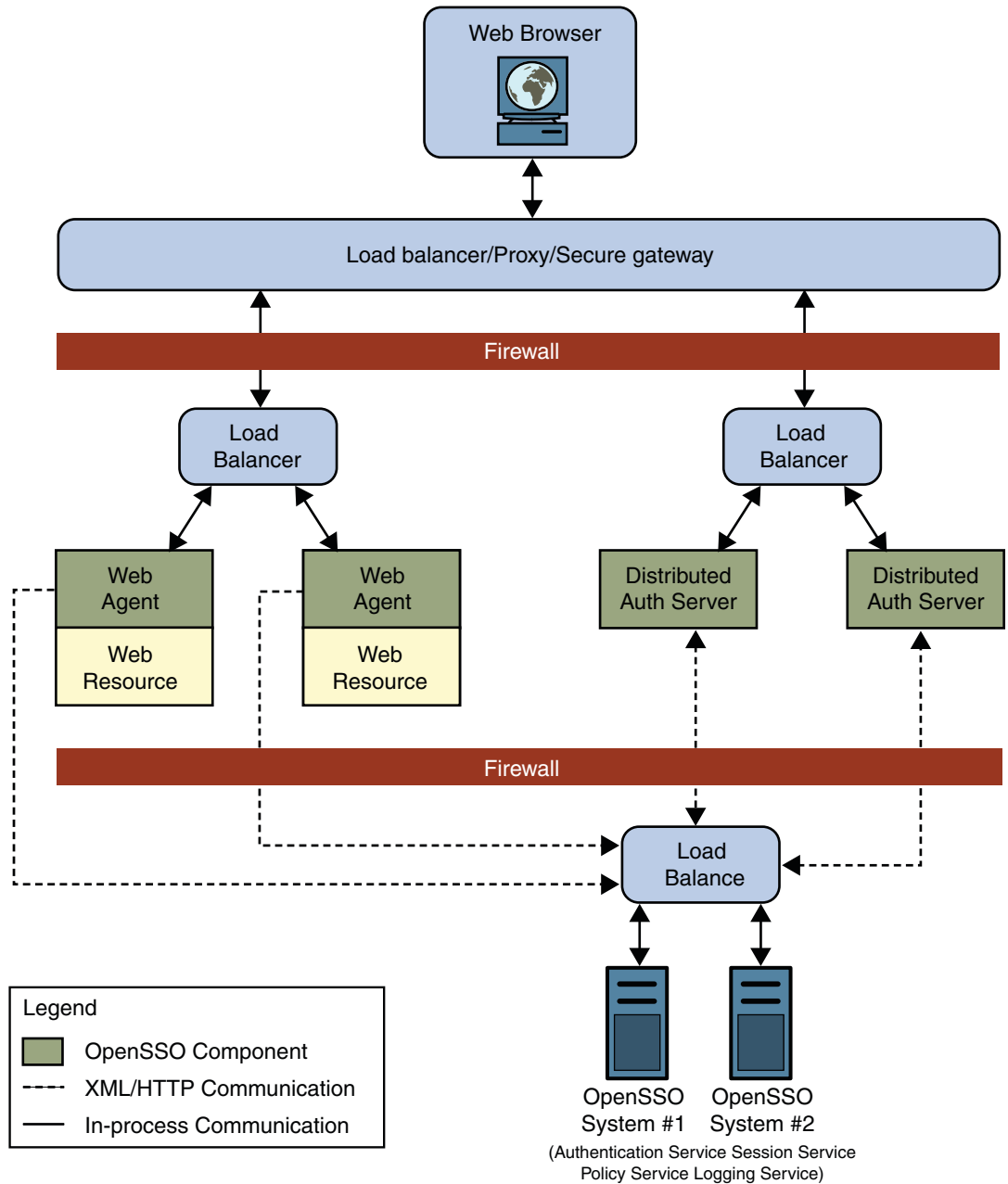


FIGURE 2-4 High Level View of an OpenSSO Enterprise Deployment With Web Agents, Firewalls, and Load Balancers

Another example of a deployment variable concerns authentication levels. In a real-world deployment, different resources on a deployment container (such as an application 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 and J2EE agents. Some of the basic steps in the policy decision process are depicted in [Figure 2-5](#). 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 2-6](#) for the J2EE agent view and [Figure 2-7](#) 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.

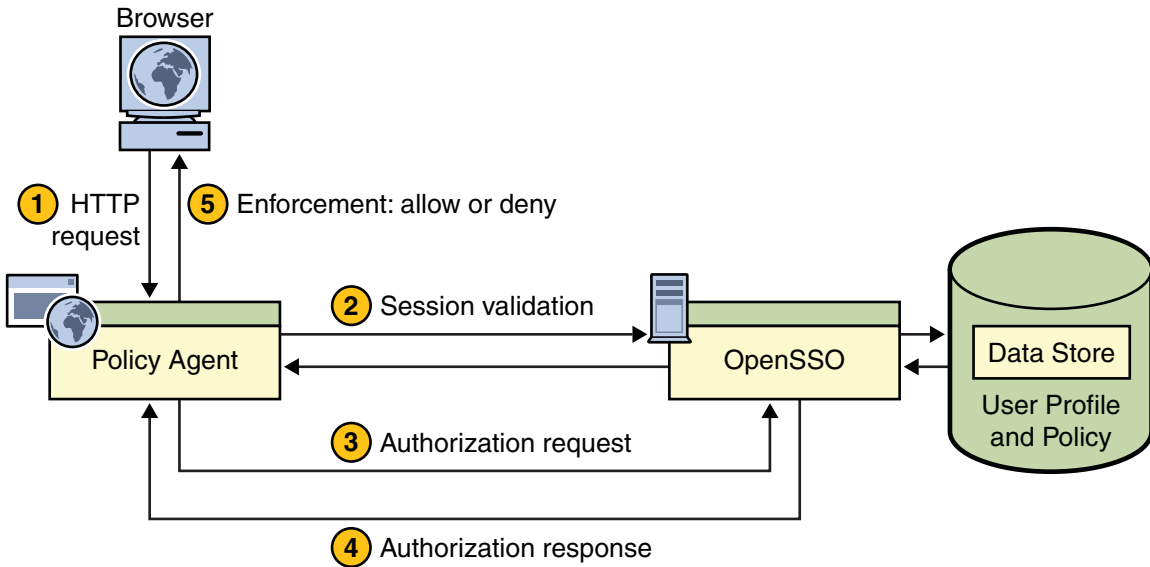


FIGURE 2-5 Policy Agent and the Policy Decision Process

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 SSO token.

As explained in preceding text, this example assumes that the user's credentials have already been authenticated. Though an SSO session such as this often would *not* require Policy Agent and OpenSSO Enterprise to contact each other during session validation, such contact is sometimes necessary and, therefore, is depicted in [Figure 2-5](#).
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.

Policy Decision Process for J2EE Agents

The figure that follows is a flow chart of the policy decision process for J2EE agents.

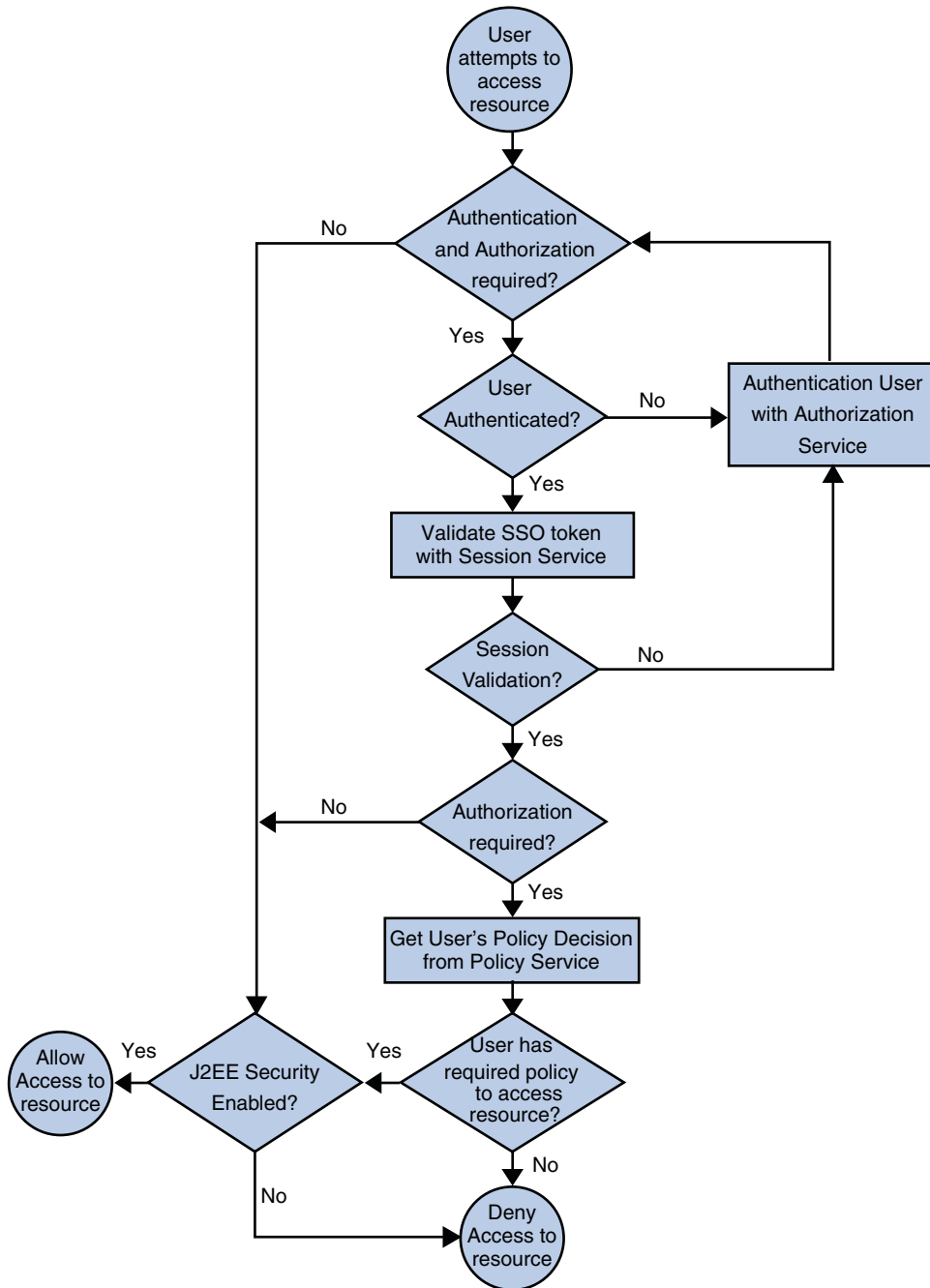


FIGURE 2-6 Policy Agent and the Policy Decision Process

Policy Decision Process for Web Agents

The figure that follows is a flow chart of the policy decision process for web agents.

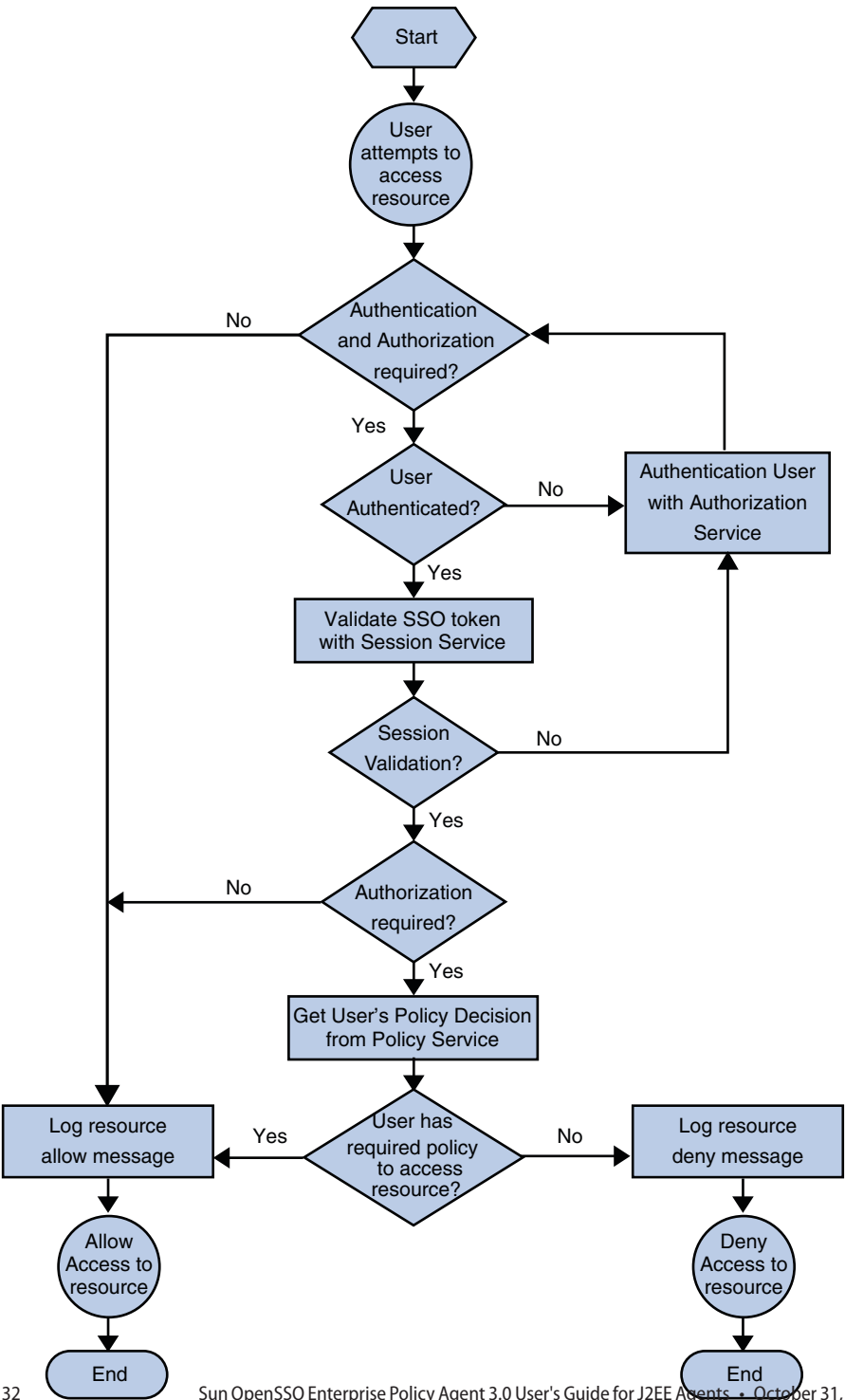


FIGURE 2-7 Policy Agent and the Policy Decision Process

Policy Agent 3.0 and the Centralized Agent Configuration Option

The option to centralize the agent configuration on the OpenSSO Enterprise server, specifically in the central data repository, is new for Policy Agent 3.0. The way this option works is the same for both web agents and J2EE agents. The following figure illustrates the use of this new feature.

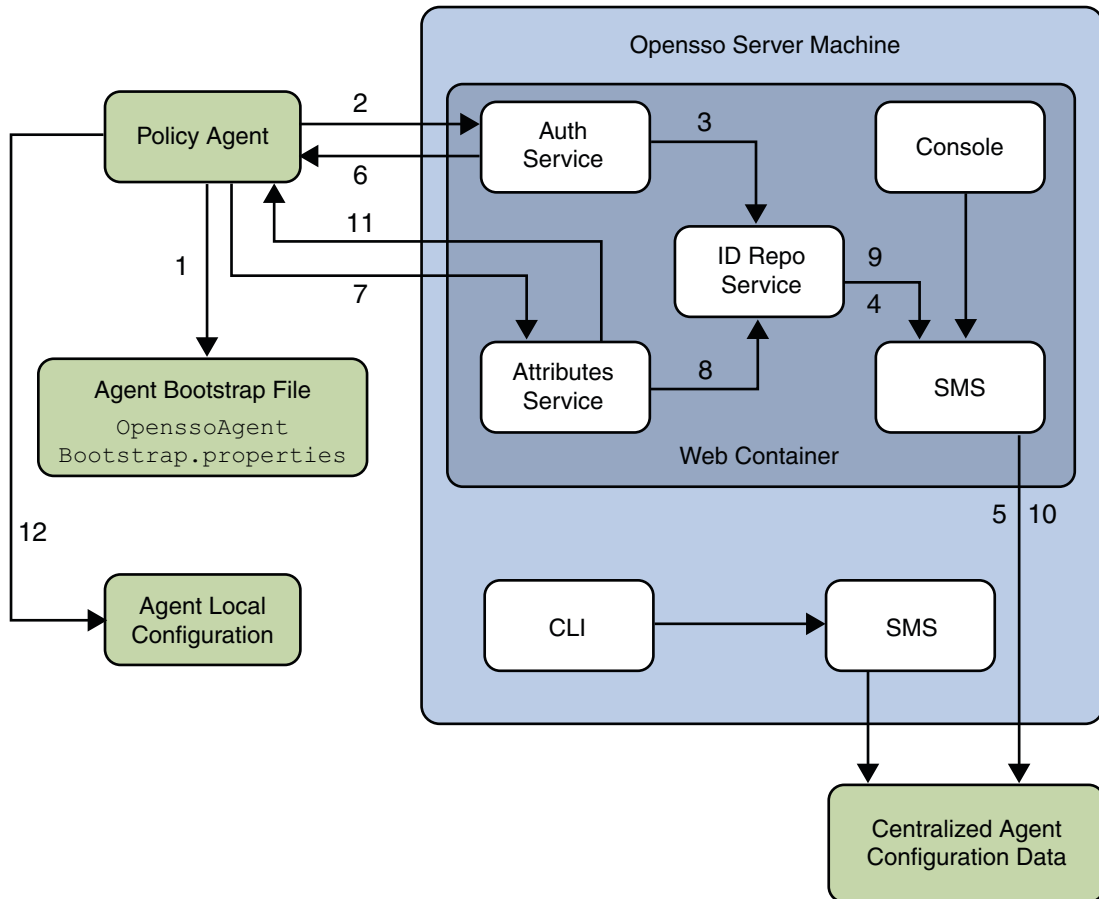


FIGURE 2-8 Centralized Agent Configuration in Policy Agent

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. For example, the LDAP service verifies that the user name and password are stored in an LDAP v3 compliant directory server, such as Sun Java System Directory Server. After the user's credentials are properly authenticated, the agent examines all the roles and 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 policy 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 2–6](#) and [Figure 2–7](#). 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. 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 the 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 manages authorization. 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 23](#) 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. The details about agent features and tasks are not provided in this section. For details about the

features and tasks for J2EE agents, see. For details about the features and tasks for web agents, see. 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 36](#)
- [“Policy Agent Log Files” on page 36](#)
- [“Not-Enforced Lists” on page 36](#)
- [“Personal Profile Attributes and Session Attributes” on page 37](#)

Configuration Properties

All agent configurations have a `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.

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 to be configured using the OpenSSO Enterprise Console or the `ssoadm` command-line utility. If the agents are 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 editing the `OpenSSOAgentConfiguration.properties` file.

Remark 2–3 Reviewer

I'm unclear about the reach of the `ssoadm` command-line utility. Can it be used to edit properties in the bootstrap file?

For the local configuration scenario, the `OpenSSOAgentConfiguration.properties` files differ slightly between web agents and J2EE agents. The biggest difference between the two agent types is that J2EE agents have extra constructs such as map constructs and list constructs. Configuration properties that are present in the files for both agent types tend to be very similar in terms of functionality.

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 URL list and a not-enforced IP Address list.

A not-enforced URL list is a list of URLs that are not protected by an agent. A resource represented by a URL on a not-enforced URL list is widely available, without restrictions. This list can be set to have a reverse meaning. With a reverse meaning, only URLs on the list are protected. All other 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 and Session Attributes

Both agent types can fetch and pass along personal profile attributes and session attributes. Client applications protected by an agent can then use information from these attributes to personalize content for the user.

Remark 2–4
Reviewer I suppose that I should mention policy response attributes here, no?

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.

Differences in Protected Resources

Web agents are capable of protecting resources that can be hosted on the web or proxy servers on which they are installed. This protection includes any resource that can be represented as a uniform resource identifier (URI) available on the protected server. Such a protected URI can be resolved by the server to static content files such as HTML files or dynamic content generation programs such as CGI scripts or servlets hosted by an embedded servlet engine. In other words, before a request is evaluated by the web or proxy server, the web agent can evaluate the necessary credentials of a user and can allow or deny access for the requested resource. Once the request is granted access to the resource, it can be processed internally by the web or proxy server as applicable. In other words, the web agent uses the request URL to enforce all policy decisions regardless of what that URL maps to internally in the web server. In cases where the request URL maps to a servlet which in turn invokes other servlets or JSPs, the web agent will not intercept these subsequent resource requests unless such invocation involves a client-side redirect.

A J2EE agent is capable of protecting web and enterprise applications hosted by the application or portal server on which it is installed. These applications may include resources such as HTML pages, servlets, JSP, and Enterprise JavaBeans (EJB). Apart from these resources, any resource that can be accessed as a URI within a protected web application can also be secured by

such agents. For example, images that are packaged within a web application can also be protected by the J2EE Policy Agent. These agents allow the evaluation of J2EE policies and can also enforce OpenSSO Enterprise based URL policies like a web agent on the resources being requested by the user. Minimally the enforcement is done at the outermost requested URL, but can also be done on any intermediate URLs being chained to this resource on most application servers.

Default Scope of Protection

When installed, a web agent automatically protects the entire set of resources available on the web server. However, in order to protect resources within a web application hosted on an application server, the web application must be 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 relatively non-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 both J2EE agents and web agents, the modes can be set in the OpenSSO Enterprise Console.

In the J2EE_POLICY and ALL modes of operation, J2EE agents enforce J2EE declarative policies as applicable for the protected application and also provide support for evaluation of programmatic security APIs available within J2EE specifications.

J2EE Agents in the Policy Agent 3.0 Release

This guide focuses on J2EE agents, the functionality of which has increased for this release. Therefore, this section provides more information about how J2EE agents function generally and what is new for the 3.0 release.

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

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

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

- “Uses of J2EE Agents” on page 39
- “How J2EE Agents Work” on page 41
- “Using the J2EE Agent Sample Application in Policy Agent 3.0” on page 42

Uses of J2EE Agents

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

The way J2EE agents function in the 3.0 release can be quite different when compared to previous releases because of the option of storing the configuration in the central data repository of the OpenSSO Enterprise server. However the purpose of J2EE agents has not changed significantly. The agents perform more effectively and efficiently in this release, but the basic purpose is the same.

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

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

J2EE Agents and an Online Auction Application

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

Remark 2–5 Reviewer

A customer once complained about the roles used in this section. He thought it was confusing to use the role "Employee" Anyway, see below that I've changed both "Employee" and "Buyer" to "Bidder." Is this a good change to make?

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

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

J2EE Agents and a Web-Based Commerce Application

A web-based commerce application may have a variety of specialized EJB components that offer a spectrum of services to clients. For instance, there could be a specialized component that enables the creation of purchase orders. Similarly, there could be a specialized component that allows the approval of purchase orders. While such components provide the basic business services for the application to function, the very nature of tasks that they accomplish requires a security policy to enforce appropriate use of such services. Using the deployment descriptors, the application vendor or developer can express this intent by protecting such components using abstract security role names. For example, a developer can create a role called Buyer to

protect the component that allows the creation of a purchase order and a role called Approver to protect the component that enables the approval of a purchase order. While these roles convey the intent of an application developer to enforce such security policies, they will not be useful unless these abstract role names are mapped to real life principals such as actual users or actual roles that reside in OpenSSO Enterprise. The J2EE agent enables the container to enforce such a runtime linkage of abstract security roles to real life principals. Once the agent is installed and configured, the application security roles can be mapped to real principals. For example, the role Buyer is mapped to an OpenSSO Enterprise role called Staff, and the role Approver is mapped to an OpenSSO Enterprise role called Manager. Thus when user Arvind tries to access the application's protected resources to create a purchase order, the agent allows this access only if this user is a member of the mapped role Staff. Similarly, a user Jamie may wish to approve this purchase order, which will be allowed by the agent only if this user is a member of the mapped role Manager.

J2EE Agents and a Content-Based Web Application

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

How J2EE Agents Work

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

- **Agent Realm**

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

- **Agent Filter**

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

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

The agent provides a fully configured and ready-to-use client installation of OpenSSO Enterprise SDK for the deployment container. This SDK offers a rich set of APIs supported by OpenSSO Enterprise that can be used to create security-aware applications that are tailored to work in the security framework offered by OpenSSO Enterprise. For more information on how to use OpenSSO Enterprise SDK, see [Sun Java System Access Manager 7.1 Developer's Guide](#).

Using the J2EE Agent Sample Application in Policy Agent 3.0

The Sample Application

Deploy, and interact with the sample application included with Policy Agent 3.0. Interacting with the sample application is perhaps the best way available to you to learn how J2EE agents work. The sample application is deployed at `URI/sampleapp`. The sample application demonstrates agent configuration options and features. With this application, you can view agent configuration examples and you can test if an agent was deployed successfully. To learn more about this application, read about the `sampleapp` directory explained in the list following [Table 3–2](#). Moreover, the `sampleapp` directory includes a `README.TXT` explaining how to build and deploy the sample application. While you can build the sample application if you desire, this step is not required. When you unpack the J2EE agent distribution, a sample application named `agentsample.ear` is created for you. The full path to this application is as follows:

PolicyAgent-base/sampleapp/dist/agentsample.ear

Vital Installation Information for a J2EE Agent in Policy Agent 3.0

To make the installation process of a J2EE agent in Policy Agent 3.0 simple, essential information needed for the installation is provided in this chapter.

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

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

- ☐ Getting the J2EE agent distribution files on the machine that hosts the deployment container. The J2EE agent is going to protect the content on that deployment container.
- ☐ Issuing install-related commands using the `agentadmin` program. The `agentadmin` program is a command-line utility that you will use to install and configure the agent. You should know the supported command options besides the more common `-install` option.
- ☐ Locating the various J2EE agent files after you get them onto the deployment container.
- ☐ Configuring the J2EE agent with OpenSSO Enterprise 6.3 Patch 1 or greater, if such backward compatibility is desired.
- ☐ Creating a J2EE agent profile.

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

- [“Format of the Distribution Files for a J2EE Agent Installation in Policy Agent 3.0” on page 44](#)
- [“Role of the `agentadmin` Program in a J2EE Agent for Policy Agent 3.0” on page 47](#)
- [“J2EE Agent Directory Structure in Policy Agent 3.0” on page 59](#)
- [“Configuring A J2EE Agent With Access Manager 7.0” on page 64](#)
- [“Creating a J2EE Agent Profile” on page 64](#)

Format of the Distribution Files for a J2EE Agent Installation in Policy Agent 3.0

The distribution files for a J2EE agent in Policy Agent 3.0 are provided to you in either a .tar.gz archive or a .zip archive. To unpack the .tar.gz archive, use the GNU_tar program.



Caution – For .tar.gz archives, do not use a program other than GNU_tar to untar the contents of the J2EE agent deliverables. Using a different program, such as another tar program, can result in some files not being extracted properly. To learn more about the GNU_tar program, visit the following web site:

<http://www.gnu.org/software/tar/tar.html>

Each J2EE agent in the Policy Agent 3.0 release presents you with three compressed-file options. The file names vary for each agent. For example, Policy Agent 3.0 for Sun Java System Application Server 9.1 provides the three compressed-file options as follows:

```
SJS_Appserver_81_agent_2.2.tar.gz
SJS_Appserver_81_agent_2.2.zip
SJS_Appserver_81_agent_2.2_SUNWamas.tar.gz
```

For other J2EE agents, the file names are different, but the naming format is the same.

Choose the file format that best fits your platform needs. The file that is in a .zip format is for Windows systems. The other two compressed files both end in .tar.gz. The .tar.gz file with SUNWam_xx in the file name contains distribution files in a package format. This package format is for Solaris systems only. The other .tar file, the one containing non-packaged distribution files, is for any UNIX-based system (including Solaris systems).

The task of unpacking the compressed file that contains the J2EE deliverable files is explained in this section separately for each platform as follows:

- “To Unpack Non-Package Formatted Deliverables of a J2EE Agent in Policy Agent 3.0” on page 45
- “To Unpack Package Formatted Deliverables of a J2EE Agent in Policy Agent 3.0” on page 45
- “To Unpack a .zip Compressed file of a J2EE Agent in Policy Agent 3.0” on page 46

Note – If you do not use the Solaris package distribution, therefore you use one of the other two formats, choose a directory in which to unpack the compressed file. Choose a directory location that best fits your needs, but note that unpacking the .zip or .tar file distributes the J2EE agent files within the directory you have chosen, creating subdirectories where necessary, but containing all of the subdirectories within the original directory that you have chosen.

▼ To Unpack Non-Package Formatted Deliverables of a J2EE Agent in Policy Agent 3.0

Follow the steps described in this task to unpack a compressed file that is named in the following manner:

appserver_version_agent_2.2.tar.gz

The two following facts apply to the this compressed file: It is for any UNIX-based system, including Solaris systems, and the distribution files contained within this compressed file are *not* in package format.

- **Issue the following command:**

```
# gzip -dc appserver_version_agent_2.2.tar.gz | tar xvf -
```

where *appserver_version* represents the specific deployment container, followed by an underscore “_”, followed by the version.

For example, the command to unpack the archive of Policy Agent 3.0 for Sun Java System Application Server 9.1 is as follows:

```
# gzip -dc SJS_Appserver_81_agent_2.2.tar.gz | tar xvf -
```

For other J2EE agents, the file names are different, but the naming format is the same.

More Information Purpose of This Task

The preceding step unpacks the archive and provides you with the agent deliverables for Policy Agent 3.0 for Sun Java System Application Server 9.1.

▼ To Unpack Package Formatted Deliverables of a J2EE Agent in Policy Agent 3.0

Follow the steps described in this task to unpack a compressed file that is named in the following manner:

appserver_version_agent_2.2_SUNWam_xx.tar.gz

The two following facts apply to the this compressed file: It is for Solaris systems only and the distribution files contained within this compressed file are in package format.

1 Issue the following command:

```
# gzip -dc appserver_version_agent_2.2_SUNWam_xx.tar.gz | tar xvf -
```

appserver_version is the name of the specific deployment container, followed by an underscore “_”, followed by the version

xx is an abbreviated name for the specific deployment container

For example, the command to unpack the archive of Policy Agent 3.0 for Sun Java System Application Server 9.1 is as follows:

```
# gzip -dc SJS_Appserver_81_agent_2.2_SUNWamas.tar.gz | tar xvf -
```

2 Issue the following command:

```
# pkgadd -d
```

By default, this command distributes the J2EE agent packages to the following directory:

```
/opt/j2ee_agents
```

More Information Purpose of This Task

The preceding steps unpack the archive and provide you with the agent deliverables for Policy Agent 3.0 for Sun Java System Application Server 9.1.

▼ To Unpack a .zip Compressed file of a J2EE Agent in Policy Agent 3.0

Follow the steps described in this task to unpack a compressed file that is named in the following manner:

```
appserver_version_agent_2.2.zip
```

The following fact applies to this compressed file: It is for Windows systems.

● **Issue the following command:**

```
unzip appserver_version_agent_2.2.zip
```

where *appserver_version* represents the specific deployment container, followed by an underscore “_”, followed by the version.

For example, the command to unpack the archive of Policy Agent 3.0 for Sun Java System Application Server 9.1 is as follows:

```
unzip SJS_Appserver_81_agent_2.2.zip
```

For other J2EE agents, the file names are different, but the naming format is the same.

More Information Purpose of This Task

The preceding step unpacks the archive and provides you with the agent deliverables for Policy Agent 3.0 for Sun Java System Application Server 9.1.

Role of the agentadmin Program in a J2EE Agent for Policy Agent 3.0

The agentadmin program is a required install and configuration tool for the 3.0 release of J2EE agents. The most basic of tasks, such as installation and uninstallation can only be performed with this tool.

The location of the agentadmin program is as follows:

PolicyAgent-base/bin

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

- All agent installation and uninstallation is 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.

Note – In this section, the options described are the agentadmin program options that apply to all J2EE agents. Options that only apply to specific J2EE agents are relatively uncommon and are described where necessary within the corresponding J2EE agent guide.

The --getUuid option as listed in the following table was not available in the original build of Agent for Sun Java System Application Server 9.1. However, the --getUuid option is available in hot patch builds of this agent.

TABLE 3-1 The agentadmin Program: Supported Options

Option	Task Performed
--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
--uninstallAll	Uninstalls all agent instances
--getUuid	Retrieves a universal ID for valid identity types
--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.

EXAMPLE 3-1 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 a J2EE 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 3-2 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 --uninstall

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

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

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

EXAMPLE 3-3 Command Format: agentadmin --uninstall (Continued)

```
./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 <i>filename</i> in command examples. The response file, or state file, can then be used for silent uninstallations. |
| --useResponse | Use this argument to uninstall a J2EE agent in silent mode as all uninstaller prompts are answered with responses previously saved to a response file, represented as <i>filename</i> in command examples. When this argument is used, the uninstaller runs in non-interactive mode. At which time, user interaction is not required. |
| <i>filename</i> | 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 3-4 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
```

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
```

EXAMPLE 3-4 Command Usage: `agentadmin --uninstall` (Continued)

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 3-5 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 3-6 Command Usage: `agentadmin --listAgents`

Issuing the `agentadmin` command with the `--listAgents` option provides you with information about all the configured J2EE agents on that machine. For example, if two J2EE agents were configured on Sun Java System Application Server 9.1, the following text demonstrates the type of output that would result from issuing this command:

The following agents are configured on this Application Server.

```
The following are the details for agent Agent_001 :-  
Application Server Config Directory:  
/var/opt/SUNWappserver/domains/domain1/config  
Application Server Instance name: server1
```

```
The following are the details for agent Agent_002 :-  
Application Server Config Directory:  
/var/opt/SUNWappserver/domains/domain1/config  
Application Server Instance name: server2
```

This example shows that two instances of the agent are configured: one for `server1` and one for `server2`. Notice that the `agentadmin` program provides unique names, such as `Agent_001` and `Agent_002`, to all the J2EE agents that protect the same instance of a deployment container, in this case Application Server 8.1. Each name uniquely identifies the J2EE agent instance.

agentadmin --agentInfo

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

EXAMPLE 3-7 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 3-8 Command Usage: agentadmin --agentInfo

Issuing the agentadmin command with the --agentInfo option provides you with information on the J2EE agent instance that you name in the command. For example, if you want information about a J2EE agent instance named Agent_002 configured on Sun Java System Application Server 9.1, you can issue the command illustrated in the following example to obtain the type of output that follows:

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

```
The following are the details for agent Agent_002 :-  
Application Server Config Directory:  
/var/opt/SUNWappserver/domains/domain1/config  
Application Server Instance name: server2
```

In the preceding example, notice that information is provided only for the agent instance, Agent_002, named in the command.

agentadmin --version

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

EXAMPLE 3-9 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 3-10 Command Usage: agentadmin --version

Issuing the agentadmin command with the --version option provides you with version information for the configured J2EE agents on that machine. For example, if a J2EE agent were configured on Sun Java System Application Server 9.1, the following text demonstrates the type of output that would result from issuing this command:

```
-----  
Sun Java(TM) System Access Manager Policy Agent for:  
Sun Java(TM) System Application Server 8.1  
-----
```

```
Version: 2.2  
Build Number: 05  
AM 70 Client SDK Version: 20050810.2  
AM 63 Client SDK Version: 20050914.1  
Date: 2005-09-15 15:04 PDT  
Build Platform: machinename
```

In the preceding example, notice that the Version field shows the major version number. The Build Number shows the minor version number. The Date field provides the date and time the agent was built, while the Build Platform field provides information about the platform on which the agent was built. The Client SDK versions signify the OpenSSO Enterprise related client SDK versions that were shipped with the agent.

agentadmin --encrypt

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

EXAMPLE 3-11 Command Format: agentadmin --encrypt

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

EXAMPLE 3–11 Command Format: agentadmin --encrypt (Continued)

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

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 a J2EE 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: **[Remark 3–3 Reviewer: I am not confident about the explanation above or the example below. Is this the correct property? Please make suggestions as appropriate.]**

```
am.encryption.pwd = EphgFhMF6X3XmMjYGCuTYHSYA9C7qQlk
```

fullpassfile Use this option to specify the full path to the password file that will be encrypted.

The password file should be created as a J2EE agent pre-installation task.

EXAMPLE 3–12 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 J2EE agent instance directory Agent_001:

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

The following is an example of an encrypted value:

```
ASEWEJIowNBjHTv1UGD324kmT==
```

Each agent uses a unique agent ID and password to communicate with OpenSSO Enterprise. Once the agent profile for a specific agent has been created in OpenSSO Enterprise, the installer assigns the Policy Agent profile name and encrypted password in the respective J2EE 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:

EXAMPLE 3-12 Command Usage: agentadmin --encrypt (Continued)

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

agentadmin --getEncryptKey

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

EXAMPLE 3-13 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 3-14 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 J2EE agent. **[Remark 3-5 Reviewer: Above, is it accurate to mention the bootstrap file and only the bootstrap file?]**

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

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

```
Agent Encryption Key : k1441g4Eeju0gsPlF0Sg+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.client.encryptionKey
```

EXAMPLE 3-14 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: **[Remark 3-6 Reviewer: I'm confused here. Is the property correct in the line of code below ?]**

```
com.sun.identity.client.encryptionKey = k1441g4Eeju0gsPLF0Sg+m6P5x7/G9rb
```

[Remark 3-7 Reviewer: For the line above, what about this property?: am.encryption.pwd. Does it apply instead?] 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 3-15 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 3-16 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 J2EE agent instances or all J2EE agent instances. 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 a J2EE agent. You can then decide, case by case, to remove a J2EE agent instance or not.

agentadmin --getUuid

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

EXAMPLE 3-17 Command Format: agentadmin --getUuid

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

```
./agentadmin --getUuid userName IdType realmName
```

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

<i>userName</i>	Use this first parameter of the --getUuid option to specify the name associated with the identity type. The identity type is represented in this example as the <i>IdType</i> parameter. Therefore, if the identity type is for a user, this <i>userName</i> parameter would be the name of that user.
<i>IdType</i>	Use this second parameter to specify a valid identity type. The following are examples of valid identity types: user, role, group, filtered role, agent, and such.
<i>realmName</i>	Use this third parameter to specify the name of the default organization of the OpenSSO Enterprise installation.

For example, if the ID of the user is manager, the identity type is role, and the realm name is dc=example,dc=com, the following would be the universal ID:

```
id=manager,ou=role,dc=example,dc=com
```



Caution – The universal ID concept is only valid starting with OpenSSO Enterprise 8.0. Do not use this option with earlier versions of OpenSSO Enterprise, such as version 6.3. If the application is deployed with OpenSSO Enterprise 6.3 principals or roles, replace the role-to-principal mappings with the distinguished name (DN) of the user in OpenSSO Enterprise 6.3.

EXAMPLE 3-18 Command Usage: agentadmin --getUuid

In OpenSSO Enterprise 8.0, issuing the agentadmin command with the --getUuid option retrieves the universal ID of any identity type in OpenSSO Enterprise 8.0.

EXAMPLE 3-18 Command Usage: agentadmin --getUuid *(Continued)*

If you run the agent in J2EE_POLICY mode, you must repackage the web applications with OpenSSO Enterprise role-to-principal mappings. The universal identifier is a way to make the name of the identity user unique.

Use the correct universal ID generated by this command in a deployment descriptor that is application container specific.

agentadmin --usage

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

EXAMPLE 3-19 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 3-20 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
```

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

The available options are:

```
--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 ID.  
--version: Displays the version information.  
--encrypt: Encrypts a given string.  
--getEncryptKey: Generates an Agent Encryption key.  
--uninstallAll: Uninstalls all the agent instances.  
--getUuid: Retrieves a universal ID for valid identity types.
```

EXAMPLE 3-20 Command Usage: `agentadmin --usage` (Continued)

```
--usage: Display the usage message.
--help: Displays a brief help message.
```

The preceding output serves as the content for the table of `agentadmin` options, introduced at the beginning of this section.

agentadmin --help

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

EXAMPLE 3-21 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 3-22 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.

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

J2EE Agent Directory Structure in Policy Agent 3.0

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

Location of the J2EE Agent Base Directory in Policy Agent 3.0

Unpacking the J2EE agent binaries creates a directory named `j2ee_agents`, within which an agent-specific directory is created. For example, if the J2EE agent being installed is Policy Agent 3.0 for Sun Java System Application Server 9.1, the directory created is named `am_as81_agent`. For other J2EE agents, the directory name is slightly different, but the naming format is the same.

This agent-specific directory is the Policy Agent base directory, referred to throughout this guide as the *PolicyAgent-base* directory. For the full path to the *PolicyAgent-base* directory, see [Example 3–23](#). For information about choosing a directory in which to unpack the J2EE agent binaries, see “[To Unpack Non-Package Formatted Deliverables of a J2EE Agent in Policy Agent 3.0](#)” on page 45.

EXAMPLE 3–23 Policy Agent Base Directory

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

Agent-HomeDirectory/j2ee_agents/am_as81_agent

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

Inside the J2EE Agent Base Directory in Policy Agent 3.0

After you finish installing an agent by issuing the `agentadmin ---install` command and interacting with the installer, you will need to access J2EE agent files in order to configure and otherwise work with the product. Within the Policy Agent base directory are various subdirectories that contain all agent configuration and log files. The structure of the Policy Agent base directory for a J2EE agent is illustrated in [Table 3–2](#).

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

PolicyAgent-base/item-name

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

PolicyAgent-base/bin

TABLE 3-2 Example of Policy Agent Base Directory for a J2EE Agent

Directory Contents: Files and Subdirectories	
LICENSE.TXT	jce
README.TXT	jsse
THIRDPARTYLICENSEREADME.TXT	lib
bin	locale
config	logs
data	sampleapp
etc	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:

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

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

PolicyAgent-base/sampleapp/readme.txt

The full path to the sample application is as follows:

PolicyAgent-base/sampleapp/dist/agentsample.ear

For more information about the sample application, see [“The Sample Application” on page 42](#).

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 a J2EE Agent for Policy Agent 3.0” on page 47](#).

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

logs	<p>This directory contains various log files, including log files created when you issue the <code>agentadmin</code> command.</p> <p>This directory also contains the installation log file. For the 3.0 release of Policy Agent, log information is stored in the installation log file after you install a J2EE agent instance. The following is the location of this log file:</p> <p><i>PolicyAgent-base/logs/audit/install.log</i></p>
lib	<p>The <code>lib</code> directory has a list of all the agent libraries that are used by the installer as well as the agent run time.</p>
locale	<p>This directory has all the agent installer information as well as agent run time specific locale information pertaining to the specific agent.</p>
data	<p>This directory has all the installer specific data.</p>



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	<p>The full path for this directory is as follows:</p> <p><i>PolicyAgent-base/AgentInstance-Dir</i></p> <p>where <i>AgentInstance-Dir</i> refers to an agent instance directory, which in this case is <code>Agent_001</code>.</p>
-----------	--

Note – This directory does not exist until you successfully install the first instance of a J2EE agent. Once you have successfully executed one run of the `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 an application 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.

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

PolicyAgent-base/Agent_001

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

Agent instance directories contain directories named config and logs.

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

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

Example of an Agent Instance (Agent_001) Directory	
logs	
config	
logs	Two subdirectories exist within this directory as follows: <div><div>audit</div><div>This directory contains the local audit trail for the agent instance.</div><div>debug</div><div>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.</div></div>
<div><div>Note – Agent-specific debug information is not stored in this directory when the J2EE agent and OpenSSO Enterprise are installed on the same deployment container. However, the J2EE agent and OpenSSO Enterprise must both support the same deployment container for this coexistence scenario to apply. When this coexistence applies, the debug information is stored in the following OpenSSO Enterprise directory:</div><div><code>/var/opt/SUNWam/debug</code></div></div>	
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.

Remark 3–8
Reviewer

Is the above description accurate?

Configuring A J2EE Agent With Access Manager 7.0

Policy Agent 3.0 was released with OpenSSO Enterprise 7 and is designed to take advantage of functionality present in this release. However, J2EE agents in the Policy Agent 3.0 release can be configured to run with OpenSSO Enterprise 6.3 Patch 1 or greater.

Certain features that Policy Agent 3.0 takes advantage of in OpenSSO Enterprise 8.0 are not available in OpenSSO Enterprise 6.3, such as “composite advices,” “policy-based response attributes,” and others.

You can configure a J2EE agent in the Policy Agent 3.0 release to communicate with OpenSSO Enterprise 6.3 Patch 1 or greater as described in the following tasks, which are divided into pre-installation, installation, and post-installation steps.

Creating a J2EE Agent Profile



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

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

You create agent profiles in OpenSSO Enterprise Console, not by configuring J2EE agent software. Creating the agent profile is a required security-related task.

The agent profile is created and modified in OpenSSO Enterprise Console. Therefore, tasks related to the agent profile are discussed in OpenSSO Enterprise documentation. Nonetheless, tasks related to the agent profile are also described in this Policy Agent guide, specifically in this section. For related information about defining the Policy Agent profile in OpenSSO Enterprise Console, see the following section of the respective document: [“Agents Profile” in Sun Java System Access Manager 7.1 Administration Guide](#).

▼ To Create an Agent Profile

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

- 1 **With the Access Control tab selected click the name of the realm for which you would like to create an agent profile.**
- 2 **Select the Subjects tab.**
- 3 **Select the Agent tab.**
- 4 **Click New.**
- 5 **Enter values for the following fields:**
 - ID.** Enter the name or identity of the agent. This is the agent profile name, which is the name the agent uses to log into Access Manager. Multi-byte names are not accepted.
 - Password.** Enter the agent password. This password must be different than the password used by the agent during LDAP authentication.
 - Password (confirm).** Confirm the password.
 - Device Status.** Select the device status of the agent. The default status is Active. If set to Active, the agent will be able to authenticate to and communicate with Access Manager. If set to Inactive, the agent will not be able to authenticate to Access Manager.
- 6 **Click Create.**
 - The list of agents appears.
- 7 **(Optional) If you desire, add a description to your newly created agent profile:**
 - a. **Click the name of your newly created agent profile from the agent list.**

b. In the Description field, enter a brief description of the agent.

For example, you can enter the agent instance name or the name of the application it is protecting.

c. Click Save.

Common J2EE Agent Tasks and Features in Policy Agent 3.0

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

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

- “Common J2EE Agent Tasks and Features” on page 67
- “Key Features and Tasks Performed With the J2EE agentadmin Program” on page 93
- “Key Features and Tasks Performed With the J2EE Agent API” on page 94

Common J2EE Agent Tasks and Features

This section focuses on features that involve setting J2EE agent property values. Assigning values to properties is the key method available for configuring agent features. The topics described in this section are typically those of greatest interest in real-world deployment scenarios. This section does not cover every property. For a list and description of all the properties.

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 the bootstrap file. To change the values in the bootstrap file, you must edit the file directly or use the `ssoadm` utility tool. For information about the `ssoadm` utility, see [Appendix C, “Using the ssoadm Command-Line Utility With Agents.”](#)

Remark 4–2
Reviewer

Above, I say one can set properties in the bootstrap file using the `ssoadm` utility. Is that true?

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

When the J2EE agent configuration is local, the Console is not available for agent configuration. Instead, you must set the J2EE agent properties using the `OpenSSOAgentConfiguration.properties` file.



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

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

PolicyAgent-base/AgentInstance-Dir/config

For more information about the Policy Agent 3.0 directory structure, see [“J2EE Agent Directory Structure in Policy Agent 3.0” on page 59](#).

The following topics are discussed in this section:

- [“Hot-Swap Mechanism in J2EE Agents” on page 68](#)
- [“J2EE Agent Properties That Are List Constructs” on page 69](#)
- [“J2EE Agent Properties That Are Map Constructs” on page 71](#)
- [“J2EE Property Configuration: Application Specific or Global” on page 72](#)
- [“J2EE Agent Filter Modes” on page 74](#)
- [“Enabling Web-Tier Declarative Security in J2EE Agents” on page 76](#)
- [“Enabling Failover in J2EE Agents” on page 81](#)
- [“Login Attempt Limit in J2EE Agents” on page 83](#)
- [“Redirect Attempt Limit in J2EE Agents” on page 83](#)
- [“Not-Enforced URI List in J2EE Agents” on page 84](#)
- [“Fetching Attributes in J2EE Agents” on page 85](#)
- [“Configuring FQDN Handling in J2EE Agents” on page 90](#)
- [“Using Cookie Reset Functionality in J2EE Agents” on page 91](#)
- [“Enabling Port Check Functionality in J2EE Agents” on page 92](#)

Hot-Swap Mechanism in J2EE Agents

Many J2EE agent properties are hot-swap enabled. The value for these keys, when altered, are dynamically loaded by the agent such that it is not necessary to restart the deployment container for these changes to take effect. However, in cases where the property is explicitly

identified as not enabled for hot-swap or in cases when the hot-swap mechanism is disabled on the system, the deployment container must be restarted for the changes to take effect.

When the agent is deployed on a deployment container where OpenSSO Enterprise has been configured, the hot-swap mechanism is disabled by default and cannot be used.

The hot-swap mechanism is controlled by the following configuration property:

```
com.sun.am.policy.config.load.interval
```

The valid values for this property is any unsigned integer including 0, which indicates the amount of time in seconds after which the agent will check for changes to the configuration. A setting of 0 disables the mechanism. By default, this mechanism is set to 0 and is, therefore, disabled.

This mechanism is primarily provided to facilitate the development and testing of your application in a controlled development or test environment. It is strongly recommended that this feature be disabled for production systems to ensure optimal utilization of system resources. Also, in a production system by disabling this feature, any accidental changes to the agent configuration will not take effect until the deployment container has been restarted.

The property that controls the hot-swap mechanism itself is hot-swap enabled. This means that if the hot-swap mechanism is enabled and you change the value of this property, the new value will take effect after the last hot-swap load interval expires. This can be therefore used to dynamically disable the entire hot-swap system. For example consider the following situation:

- The deployment container is started with the load interval set to 10 seconds. Therefore, changes made to the agent configuration are picked up by the agent every 10 seconds.
- If you modify the load interval value while the deployment container is running and set it to 0, when the last load interval completes, the agent will pick up this new value. Since the value is set to 0 the agent will disable the hot-swap mechanism for the entire system.
- Once disabled, the property changes made will not be sensed by the agent. Therefore, even if you reset the value of this property now to any other number, it will not enable the hot-swap mechanism unless the deployment container is restarted.

When the value of the load interval is set to 0 during the startup of the deployment container, the hot-swap mechanism will be disabled and cannot be enabled without restarting the server and ensuring that this value is set to a value greater than 0.

J2EE Agent Properties That Are List Constructs

Certain J2EE properties are specified as lists. Knowledge of the format of these list constructs is often *not* required in order to set them. For example when you configure the properties using the OpenSSO Enterprise Console, you do not interact with the “<key>[<index>] = <value>”

formatting involved with list constructs. However, if you use the 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 4-1 Example of J2EE Agent Properties That Are List Constructs

```
com.sun.am.policy.example.list[0] = value0
com.sun.am.policy.example.list[1] = value1
com.sun.am.policy.example.list[2] = value2
```

J2EE Agent Properties That Are Map Constructs

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

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

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

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

Certain J2EE agent properties are specified as maps. A map construct has the following format (Does not apply when the 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 4-2 Example of J2EE Agent Properties That Are Map Constructs

```
com.sun.am.policy.example.map[AL] = ALABAMA
com.sun.am.policy.example.map [AK] = ALASKA
com.sun.am.policy.example.map [AZ] = ARIZONA
```

J2EE Property Configuration: Application Specific or Global

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

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

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

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

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

TABLE 4-3 Use of the Map Construct Format: Required or Not *(Continued)*

Using the	Local agent configuration	YES
OpenSSOAgentConfiguration.properties file		

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

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

key	The configuration key (name of the configuration property)
appname	The application name to which this configuration belongs. The application name is the context path of the application without the leading forward slash character. In case when the application has been deployed at the root-context of the server, the application name should be specified as DefaultWebApp.
value	The value used by the agent to protect the application identified by the given application name

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

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

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

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

EXAMPLE 4-3 Example of Application Specific and Global Configuration

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

J2EE Agent Filter Modes

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

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

- NONE
- SSO_ONLY
- J2EE_POLICY
- URL_POLICY
- ALL

Regardless of what mode the agent filter is operating in, the agent realm will continue to function, if configured. This can therefore lead to a situation where the agent realm component may malfunction or may result in the negative evaluation of J2EE security policies configured in the application's deployment descriptors or being used through the J2EE programmatic security API. To avoid this, you may disable the agent realm component, if necessary. The sections that follow describe the different agent filter modes and also tell you how to disable the agent realm.

J2EE Agent Filter Mode-NONE

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

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

Although this mode disables the agent filter from taking any action on the incoming requests other than logging, it has no effect on the agent realm that may still be configured in your deployment container and may get invoked by the deployed application if the deployed application has J2EE security policies in its descriptors or uses programmatic security. With the agent filter disabled, these applications will fail to evaluate the J2EE security policies correctly and as a result the deployed application may malfunction. In order to fully disable the agent, you must therefore ensure that the agent realm is not active. Refer to the section Disabling the Agent Realm to find out how the agent realm can be disabled for your agent installation. Once you have disabled the agent realm and the filter mode is set to NONE, it is functionally equivalent to not having the agent in your system at all.

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

J2EE Agent Filter Mode - SSO_ONLY

This is the least restrictive mode of operation for the agent filter. In this mode, the agent simply ensures that all users who try to access protected web resources are authenticated using OpenSSO Enterprise Authentication Service. In this mode of operation the agent realm is not used.

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

J2EE Agent Filter Mode - J2EE_POLICY

In this mode, the agent filter and agent realm work together with various OpenSSO Enterprise services to ensure the correct evaluation of J2EE policies. These policies may be configured using the declarative security in the application's deployment descriptors, or may be implicit in the code of the application in the cases where it uses the J2EE programmatic security APIs. No URL policies defined in OpenSSO Enterprise take effect in this mode of filter operation. When the deployed application uses declarative security in the web-tier, you must configure the agent to enable this feature. See [“Enabling Web-Tier Declarative Security in J2EE Agents” on page 76](#) for more information on how to enable this feature. When running in the J2EE_POLICY mode, the agent ensures that the security principal is set in the system for every authorized user access. In the J2EE_POLICY mode, the agent will not enforce any applicable URL policies as defined in OpenSSO Enterprise.

J2EE Agent Filter Mode - URL_POLICY

In this mode, the agent filter is used to enforce various URL policies that may be defined in OpenSSO Enterprise. This mode does not require the agent realm to be functional.

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

J2EE Agent Filter Mode - ALL

This is the most restrictive mode of the agent filter. In this mode, the filter enforces both J2EE policies and URL policies as defined in OpenSSO Enterprise. This mode of operation requires

that the agent realm be configured in the deployment container. When running in the ALL mode, the agent ensures that the security principal is set in the system for every authorized access.

This mode of operation is, with very few exceptions, the preferred mode for deployed production systems.

Enabling Web-Tier Declarative Security in J2EE Agents

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

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

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

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

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

▼ To Enable J2EE Agents to Handle Security Constraint Settings

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

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

2 The form-login-page element of form-login-config should be added as one of the values for the following agent property:

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

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

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

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

`/Portal/jsp/login.jsp`

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

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

`/BankApp/SignOn`

`/ERP/LoginServlet`

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

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

Note –

- When a protected application is configured for web-tier declarative security handling by the agent, it must be redeployed with a form-login configuration as described in this section. This configuration requires that two application resources be specified in the application's `web.xml` deployment descriptor: one for the `form-login-page` and the other for the `form-error-page`. Regardless of whether the resource corresponding to the `form-login-page` exists in the application or not (this depends on how the agent is configured to handle the form-login requests), the resource corresponding to the `form-error-page` must be present in the application. This resource is directly invoked by the deployment container to indicate authentication failures and, optionally, authorization failures. If the application does not contain a valid `form-error-page` matching the URI specified in this deployment descriptor, it could result in HTTP 404 errors when the container chooses to display this error page.
- For applications that do not contain a `form-login-page`, you can specify any URI as long as that URI does not conflict with any application resource and the matching value has been added to the configuration property `com.sun.identity.agents.config.login.form`.
- By default, the agent is configured to intercept all form-login requests and handle them without invoking the actual `form-login-page` resource as specified in the `web.xml` of the protected application. Thus, when using a default installation of the agent, the application is not required to have a resource corresponding to the `form-login-page` element specified in `web.xml`. This allows for the configuration of web-tier declarative security for applications that were not designed to use the form-login mechanism and instead relied on other login schemes available in J2EE specification. This behavior of the agent can be changed so that it allows the form-login requests to be handled by actual resources that exist within the application by changing the agent configuration properties as applicable. For details on how this can be done, please refer to the section [“Customizing Agent Response for Form Login” on page 79](#).
- If the agent filter is operating in the `URL_POLICY` mode, any necessary URL policies to allow access to the `form-error-page` resource must be created for all users.

To further customize the behavior of the application when using web-tier declarative security, see [“Web-Tier Security Details” on page 78](#).

Web-Tier Security Details

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

user name as `j_username` and a valid password as `j_password` to the special URI `j_security_check` using the HTTP POST method of form submission.

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

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

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

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

Customizing Agent Response for Form Login

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

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

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

▼ To Customize the Agent Response to Form Login

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

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

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

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

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

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

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

This can be done by enabling the property labeled `User Internal Login` (`com.sun.identity.agents.config.login.use.internal`).

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

The following JSP fragment demonstrates how this can be done:

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


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

Enabling Failover in J2EE Agents

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

▼ To Enable Failover in J2EE Agents

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

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

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

primary-OSSO-server

failover-OSSO-server1

failover-OSSO-server2

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

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

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

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

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

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

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

Case 1: The Login URL Prioritized property is enabled.

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

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

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

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

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

**Remark 4–7
Reviewer**

What's with this `com.iplanet.am.naming.url`? Does it not exist in 3.0. Should another property be used instead or should this write up be completely different?

This can be done by using the following property:

```
com.iplanet.am.naming.url
```

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

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

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

Configure the following property to create the list:

```
com.sun.identity.agents.config.cdsso.cdcservlet.url[]
```

Specify more than one CDSSO URL in the following manner:

```
com.sun.identity.agents.config.cdsso.cdcservlet.url[0] = primary-remoteAM-server
```

```
com.sun.identity.agents.config.cdsso.cdcservlet.url[1] = failover-remoteAM-server1
```

```
com.sun.identity.agents.config.cdsso.cdcservlet.url[2] = failover-remoteAM-server2
```

Login Attempt Limit in J2EE Agents

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

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

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

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

Redirect Attempt Limit in J2EE Agents

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

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

The Redirect Attempt Limit configuration property allows you to guard against such potential situations by ensuring that after a given number of consecutive requests from a particular user that result in the same exact redirect, the agent blocks the user request. This blocking of the request is only temporary and is removed the moment the user makes a request that does not result in the same redirect or results in access being granted to the protected resource. The configuration property that controls this feature is:

```
com.sun.identity.agents.config.redirect.attempt.limit
```

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

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

Not-Enforced URI List in J2EE Agents

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

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

```
/images/*
```

```
/public/*.html
```

```
/registration/*
```

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

Inverting the Not-Enforced URI List

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

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



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

Note –

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

Fetching Attributes in J2EE Agents

Certain applications rely on the presence of user-specific profile information in some form in order to process the user requests appropriately. J2EE agents provide the functionality that can

help such applications by making these attributes from the user's profile available in various forms. Policy Agent 3.0 allows the following attribute types to be fetched using the corresponding properties:

Profile Attributes

```
com.sun.identity.agents.config.profile.attribute.fetch.mode
```

Session Attributes

```
com.sun.identity.agents.config.session.attribute.fetch.mode
```

Policy Response Attributes

```
com.sun.identity.agents.config.response.attribute.fetch.mode
```

The following values are possible for these three properties:

- NONE
- HEADER
- REQUEST_ATTRIBUTE
- COOKIE

The default value for these properties is `NONE`, which specifies that that particular attribute type (profile attribute, session attribute, or policy response attribute) is not fetched. The other possible values (`HEADER`, `REQUEST_ATTRIBUTE`, or `COOKIE`) that can be used with these properties specify which method will be used to fetch a given attribute type. For more information, see [“Methods for Fetching Attributes in J2EE Agents” on page 88](#).

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

The final subsection in this section describes other J2EE agent properties that can influence the attribute fetching process, see [“Common Attribute Fetch Processing Related Properties” on page 89](#).

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

Fetching Profile Attributes in J2EE Agents

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

Assigning a Mode to Profile Attributes

```
com.sun.identity.agents.config.profile.attribute.fetch.mode =  
REQUEST_ATTRIBUTE
```

The key is the profile attribute name and the value is the name under which that attribute will be made available.

Mapping Profile Attributes

```
com.sun.identity.agents.config.profile.attribute.mapping[cn]=CUSTOM-Common-Name
com.sun.identity.agents.config.profile.attribute.mapping[mail]=CUSTOM-Email

com.sun.identity.agents.config.profile.attribute.fetch.mode = REQUEST_ATTRIBUTE
com.sun.identity.agents.config.profile.attribute.mapping[] =
```

Fetching Session Attributes in J2EE Agents

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

Assigning a Mode to Session Attributes

```
com.sun.identity.agents.config.session.attribute.fetch.mode = REQUEST_ATTRIBUTE
```

The key is the session attribute name and the value is the name under which that attribute will be made available.

Mapping Session Attributes

```
com.sun.identity.agents.config.session.attribute.mapping[UserToken]=CUSTOM-userid

com.sun.identity.agents.config.session.attribute.fetch.mode = REQUEST_ATTRIBUTE
com.sun.identity.agents.config.session.attribute.mapping[] =
```

Fetching Policy Response Attributes in J2EE Agents

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

Assigning a Mode to Policy Response Attributes

```
com.sun.identity.agents.config.response.attribute.fetch.mode =  
REQUEST_ATTRIBUTE
```

The key is the policy response attribute name and the value is the name under which that attribute will be made available.

Mapping Policy Response Attributes

```
com.sun.identity.agents.config.response.attribute.mapping  
  
com.sun.identity.agents.config.response.attribute.fetch.mode =  
REQUEST_ATTRIBUTE  
com.sun.identity.agents.config.response.attribute.mapping[] =
```

Using this property for mapping policy response attributes, you can specify any number of attributes that are required by the protected application. For example, if the application requires the attributes `cn` and `mail`, and it expects these attributes to be available under the names `COMMON_NAME` and `EMAIL_ADDR`, then your configuration setting would be as follows:

```
com.sun.identity.agents.config.response.attribute.mapping[cn] = COMMON_NAME  
  
com.sun.identity.agents.config.response.attribute.mapping[mail] = EMAIL_ADDR
```

Methods for Fetching Attributes in J2EE Agents

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

- HTTP Headers
- Request Attributes
- Cookies

Fetching Attributes as HTTP Headers

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

```
long getDateHeader(java.lang.String name)  
  
java.lang.String getHeader(java.lang.String name)  
  
java.util.Enumeration getHeaderNames()  
  
java.util.Enumeration getHeaders(java.lang.String name)  
  
int getIntHeader(java.lang.String name)
```


The property that controls the parsing of a date value from an appropriate string as set in the LDAP attribute is the following:

```
com.sun.identity.agents.config.attribute.date.format
```

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

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

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

Fetching Attributes as Request Attributes

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

Fetching Attributes as Cookies

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

Multi-valued attributes are set as a single cookie value in a manner that all values of the attribute are concatenated into a single string using a separator character that can be specified by the following configuration entry:

```
com.sun.identity.agents.config.attribute.cookie.separator
```

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

When you are fetching attributes as cookies, also use the cookie reset functionality to ensure that these cookies get cleaned up from the client browser when the client browser's session expires. For more information, see [“Using Cookie Reset Functionality in J2EE Agents” on page 91](#).

Common Attribute Fetch Processing Related Properties

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

```
com.sun.identity.agents.config.attribute.cookie.separator
```

This property allows you to assign a character to be used to separate multiple values of the same attribute when it is being set as a cookie. This property is set in the following manner:

```
com.sun.identity.agents.config.attribute.cookie.separator = |
```

```
com.sun.identity.agents.config.attribute.cookie.encode
```

This property is a flag that indicates if the value of the attribute should be URL encoded before being set as a cookie. This property is set in the following manner:

```
com.sun.identity.agents.config.attribute.cookie.encode = true
```

```
com.sun.identity.agents.config.attribute.date.format
```

This property allows you to set the format of date attribute values to be used when the attribute is set to HTTP header. This format is based on the definition as provided in `java.text.SimpleDateFormat`. This property is set in the following manner:

```
com.sun.identity.agents.config.attribute.date.format = EEE, d MMM yyyy hh:mm:ss z
```

Configuring FQDN Handling in J2EE Agents

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

```
com.sun.identity.agents.config.fqdn.check.enable
```

Enables FQDN

```
com.sun.identity.agents.config.fqdn.default
```

Stores the default FQDN value

```
com.sun.identity.agents.config.fqdn.mapping[]
```

Sets FQDN mapping

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

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

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

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

In this case, assuming that `www.externalhostname.com` is the default FQDN, then the FQDN Map can be configured as follows to allow access to the application for users who will use the hostname `internalhostname.interndomain.com` or the raw IP address, say `192.101.98.45`:

```
com.sun.identity.agents.config.fqdn.mapping [internalhostname.interndomain.com] =
internalhostname.interndomain.com
```

```
com.sun.identity.agents.config.fqdn.mapping [192.101.98.45] = 192.101.98.45
```

Using Cookie Reset Functionality in J2EE Agents

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

```
com.sun.identity.agents.config.cookie.reset.enable = false
com.sun.identity.agents.config.cookie.reset.name[0] =
com.sun.identity.agents.config.cookie.reset.domain[] =
com.sun.identity.agents.config.cookie.reset.path[] =
```

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

The `com.sun.identity.agents.config.cookie.reset.name` property specifies a list of cookie names that will be reset by the agent when necessary. Each entry in this list can correspond to a maximum of one entry in the

`com.sun.identity.agents.config.cookie.reset.domain` property and the `com.sun.identity.agents.config.cookie.reset.path` property, both of which are used to define the cookie attributes - the domain on which a particular cookie should be set and the path on which it will be set.

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

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

Enabling Port Check Functionality in J2EE Agents

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

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

This situation can be controlled by enabling port check functionality on the agent. This is controlled by the following configuration property:

```
com.sun.identity.agents.config.port.check.enable
```

When this property is set to true, the agent verifies the correctness of the port number read from the request against its configuration. The configuration that provides the reference for this checking is set by the following property:

```
com.sun.identity.agents.config.port.check.setting
```

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

When the agent discovers an invalid port in the request, it takes corrective action by sending some HTML data to break the redirection chain so that the browser can reset its HOST header on the subsequent request. This content is read from the file that resides in the `locale` directory of agent installation. The name of the file is controlled by the following property:

```
com.sun.identity.agents.config.port.check.file
```

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

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

Key Features and Tasks Performed With the J2EE agentadmin Program

The agentadmin program is a utility used to perform a variety of tasks from required tasks, such as installation to optional tasks, such as displaying version information. This section summarizes the tasks that can be performed with the agentadmin program. Many of the tasks performed with this program are related to installation or uninstallation. For detailed information about the options available with this program, see [“Role of the agentadmin Program in a J2EE Agent for Policy Agent 3.0” on page 47](#).

In this section, the options are listed for your quick review to help you get a sense of how the agentadmin program fits in with the other methods of managing J2EE agents, which are all discussed in this chapter.

The location of the agentadmin program is as follows:

PolicyAgent-base/bin

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.

Note – In this section, the options described are the agentadmin program options that apply to all J2EE agents. Options that only apply to specific J2EE agents are relatively uncommon and are described where necessary within the corresponding J2EE agent guide.

The --getUuid option as listed in the following table was not available in the original build of Agent for Sun Java System Application Server 9.1. However, the --getUuid option is available in hot patch builds of this agent.

TABLE 4-4 The agentadmin Program: Supported Options

Option	Task Performed
--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

TABLE 4-4 The agentadmin Program: Supported Options (Continued)

Option	Task Performed
--getEncryptKey	Generates an Agent Encryption key
--uninstallAll	Uninstalls all agent instances
--getUuid	Retrieves a universal ID for valid identity types
--usage	Displays the usage message
--help	Displays a brief help message

Key Features and Tasks Performed With the J2EE Agent API

The agent runtime provides access to all the OpenSSO Enterprise application program interfaces (API) that can be used to further enhance the security of your application. Besides the OpenSSO Enterprise API, the agent also provides a set of API that allow the application to find the SSO token string associated with the logged-in user. These API can be used from within the web container or the EJB container of the deployment container. These are agent utility API. However, an equally viable option is to use client SDK public API directly to fetch the SSO token.

Note – Certain containers, such as Apache Tomcat Servlet/JSP Container do not have an EJB container. Hence, the EJB related agent API would not be applicable for such containers.

The subsections that follow illustrate the available agent API that can be used from within an application. The J2EE agent API have changed in Policy Agent 3.0 as explained in this section. This section includes an example of the new API in use, see [“Usage of New J2EE Agent API in Policy Agent 3.0” on page 96](#).

Class AmFilterManager

`com.sun.identity.agents.filter.AmFilterManager`

Available API for Class AmFilterManager

- `public static com.sun.identity.agents.filter.AmSSOCache
getAmSSOCacheInstance() throws com.sun.identity.agents.arch.AgentException`

Note – Deprecated: This method has been deprecated. The best practice is not to use this method, but to use the new public API for this `AmFilterManager` class as follows:

```
public static com.sun.identity.agents.filter.IAmSSOCache getAmSSOCache()
```

This method returns an instance of Class `AmSSOCache`, which can be used to retrieve the SSO token for the logged-in user. This method can throw `AgentException` if an error occurs while processing this request.

- `public static com.sun.identity.agents.filter.IAmSSOCache getAmSSOCache()`

This method returns an instance of `IAmSSOCache` interface, which can be used to retrieve the SSO token for the logged-in user.

Interface `IAmSSOCache`

`com.sun.identity.agents.filter.IAmSSOCache`

Available API for Interface `IAmSSOCache`

```
public String getSSOTokenForUser(Object ejbContextOrServletRequest)
```

This method can be used to retrieve the SSO token for the logged-in user. If called from the web tier, this method passes an instance of `javax.servlet.http.HttpServletRequest` as an argument. If called from the EJB tier, this method passes an instance of `javax.ejb.EJBContext` as an argument. This method eradicates the need to use two separate methods in `AmSSOCache` to retrieve the SSO token.

Class `AmSSOCache`

`com.sun.identity.agents.filter.AmSSOCache`

Note – Deprecated: This class and its methods have been deprecated. The best practice is not to use the methods in this class, but to use the unified API in `com.sun.identity.agents.filter.IAmSSOCache`.

Available API for Class `AmSSOCache`

- `public java.lang.String
getSSOTokenForUser(javax.servlet.http.HttpServletRequest request)`

Note – Deprecated: This method has been deprecated as explained in the Note in “[Class AmSSOCache](#)” on page 95.

This method returns the SSO token for the logged-in user whose request is currently being processed in the web container within the deployment container. This method can return null if the requested token is not available at the time of this call.

- `public java.lang.String getSSOTokenForUser(javax.ejb.EJBContext context)`

Note – Deprecated: This method has been deprecated as explained in the Note in “[Class AmSSOCache](#)” on page 95.

This method returns the SSO token for the logged on user whose request is currently being processed in the deployment container’s EJB tier. This method can return null if the requested token is not available at the time of this call.

Note – The API `getSSOTokenForUser(javax.ejb.EJBContext)` can be used only when the agent operation mode is either `J2EE_POLICY` or `ALL`.

Usage of New J2EE Agent API in Policy Agent 3.0

The following example demonstrates the new J2EE agent API in use.

EXAMPLE 4–4 Usage of New J2EE Agent API

- Web Tier Use Case:

```
String sstoken =  
    AmFilterManager.getAmSSOCache().getSSOTokenForUser(HTTPRequest);
```

- EJB Tier Use Case:

```
String sstoken =  
    AmFilterManager.getAmSSOCache().getSSOTokenForUser(EJBContext);
```



Caution – This public API can only retrieve the `SSOToken` object in EJB context if the value of the following property labeled `User Principal Flag` (`com.sun.identity.agents.config.user.principal`) is enabled

Wildcard Matching in Policy Agent 3.0 J2EE Agents

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

Remark A-1
Reviewer

Is this write up accurate? Does everything in this wildcard section fit the agent type?

- “The Multi—Level Wildcard: *” on page 98
- “The One-Level Wildcard: -*” on page 99

These wildcards can be used in policy related situations. For example, when using the OpenSSO Enterprise Console or the `ssoadm` utility to create policies or when configuring the Policy Agent property to set 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 C, “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/*` and

`http://agentHost:8090/agentsample/example-*-/example.html`. For the not-enforced list, the following are feasible examples of the wildcards in use: `/agentsample.com/*.gif` and `/agentsample.com/*-/images`.

Note – 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.

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 asterisk wildcard character when used with the forward slash (/) as the delimiter character:

- The asterisk (*) matches zero or more characters, except the question mark, in the resource name, including the forward slash (/). For example, ...B-example/* matches ...B-example/b/c/d, but doesn't match ...B-example/?
- Multiple consecutive forward slash characters (/) do not match with a single forward slash character (/). For example, ...B-example/*/A-example doesn't match ...B-example/A-example.
- Any number of trailing forward slash characters (/) are not recognized as part of the resource name. For example, ...B-example/ or ...B-example// are treated the same as ...B-example.

TABLE A-1 Examples of the Asterisk (*) as the Multi-Level Wildcard

Pattern	Matches	Does Not Match
http://A-example.com:80/*	http://A-example.com:80 http://A-example.com:80/ http://A-example.com:80/index.html http://A-example.com:80/x.gif	http://B-example.com:80/ http://A-example.com:8080/index.html http://A-example.com:80/a?b=1
http://A-example.com:80/*.html	http://A-example.com:80/index.html http://A-example.com:80/pub/ab.html http://A-example.com:80/pri/xy.html	http://A-example.com/index.html http://A-example.com:80/x.gif http://B-example.com/index.html
http://A-example.com:80/*/ab	http://A-example.com:80/pri/xy/ab/xy/ab http://A-example.com:80/xy/ab	http://A-example.com/ab http://A-example.com/ab.html http://B-example.com:80/ab

TABLE A-1 Examples of the Asterisk (*) as the Multi-Level Wildcard (Continued)

Pattern	Matches	Does Not Match
http://A-example.com:80/ab/*de	http://A-example.com:80/ab/123/de http://A-example.com:80/ab/ab/de http://A-example.com:80/ab/de/ab/de http://A-example.com:80/ab//de	http://A-example.com:80/ab/de http://A-example.com:80/ab/de http://B-example.com:80/ab/de/ab/de

The One-Level Wildcard: -*-

The one-level wildcard (-*-) matches only the defined level from the point where the one-level wildcard lies to the next delimiter boundary. The “defined level” refers to the area between delimiter boundaries. Many of the rules that apply to the asterisk 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 asterisk, 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-example/-*- doesn't match ...B-example/b/c/ or ...B-example/b?
- Multiple consecutive forward slash characters (/) do not match with a single forward slash character (/). For example, ...B-example/-*-/A-example doesn't match ...B-example/A-example.
- Any number of trailing forward slash characters (/) are not recognized as part of the resource name. For example, ...B-example/ or ...B-example// are treated the same as ...B-example.

TABLE A-2 Examples of the One—Level Wildcard (-*-)

Pattern	Matches	Does Not Match
http://A-example.com:80/b/-*-	http://A-example.com:80/b http://A-example.com:80/b/ http://A-example.com:80/b/cd/	http://A-example.com:80/b/c?d=e http://A-example.com:80/b/cd/e http://A-example.com:8080/b/
http://A-example.com:80/b/-*-/f	http://A-example.com:80/b/c/f http://A-example.com:80/b/cde/f	http://A-example.com:80/b/c/e/f http://A-example.com:80/f/
http://A-example.com:80/b/c/-*-/f	http://A-example.com:80/b/cde/f http://A-example.com:80/b/cd/f http://A-example.com:80/b/c/f	http://A-example.com:80/b/c/e/f http://A-example.com:80/b/c/ http://A-example.com:80/b/c/fg

Precautions Against Session-Cookie Hijacking in an OpenSSO Enterprise Deployment

This technical note provides information about precautions you can take against specific security threats related to session-cookie hijacking in an OpenSSO Enterprise deployment. A common concern for administrators who want to restrict access to web-based applications in an OpenSSO Enterprise deployment is that hackers might use applications, referred to as “rogue” or “untrusted,” to hijack session cookies. This document describes the threat posed by this hacking technique and provides configuration steps you can perform to guard against this threat, as described in the following sections:

- [“Defining Key Cookie Hijacking Security Issues” on page 101](#)
- [“OpenSSO Enterprise Solution: Cookie Hijacking Security Issues” on page 105](#)
- [“Implementing the OpenSSO Enterprise Solution for Cookie Hijacking Security Issues” on page 108](#)

The tasks presented in this document apply to a deployment with the following components:

- Starting with OpenSSO Enterprise 7.0
- Starting with Policy Agent 3.0

The Policy Agent 3.0 software set includes two types of agents: J2EE agents and web agents. Configuring these two agent types requires slightly different instructions, which are provided where appropriate.

Defining Key Cookie Hijacking Security Issues

The tasks presented in this document address specific security issues related to session-cookie hijacking. This section defines those security issues.

The term “cookie hijacking” simply refers to a situation where an impostor (a hacker, perhaps using an untrusted application) gains unauthorized access to cookies. Therefore, cookie hijacking, by itself, does not refer to unauthorized access to protected web resources. When the

cookies being hijacked are session cookies, then cookie hijacking potentially increases the threat of unauthorized access to protected web resources, depending upon how the system is configured.

This section provides background information about specific security issues related to session-cookie hijacking, illustrating how this type of cookie hijacking is possible and how it can potentially lead to unauthorized access of protected web resources. This section provides the underlying basis for understanding how the tasks presented in this document enable OpenSSO Enterprise to handle the specified security issues.

OpenSSO Enterprise provides Single Sign-On (SSO) and Cross Domain Single Sign-On (CDSSO) features, enabling users to seamlessly authenticate across multiple web-based applications. OpenSSO Enterprise is able to provide this seamless authentication by using hypertext transfer protocol (HTTP) or secure hypertext transfer protocol (HTTPS) to set session cookies on users' browsers.

The way OpenSSO Enterprise is configured influences the way it sets the session cookies. Prior to the implementation of the tasks outlined in this document, OpenSSO Enterprise sets session cookies for the entire domain. Therefore, all applications hosted on the domain share the same session cookies. This scenario could enable an untrusted application to intervene and gain access to the session cookies. Specific configuration steps presented in this document address this issue. After you perform the configuration, OpenSSO Enterprise prevents different applications from sharing the same session cookies.

The following list provides some details about possible security issues related to session-cookie hijacking (labels, such as “Security Issue: Insecure Protocol,” are used to make the issues easy to identify and discuss):

Security Issue: Insecure Protocol

An application does not use a secure protocol, such as HTTPS, which makes the session cookie prone to network eavesdropping.

The following security issues could apply if you do not perform the tasks presented in this document.

Security Issue: Shared Session Cookies

All applications share the same HTTP or HTTPS session cookie. This shared session-cookie scenario enables hackers to intervene by using an untrusted application to hijack the session cookie. With the hijacked session cookie, the untrusted application can impersonate the user and access protected web resources.

Security Issue: A Less Secure Application

If a single “less secure” application is hacked, the security of the entire infrastructure is compromised.

Security Issue: Access to User Profile Attributes

The untrusted application can use the session cookie to obtain and possibly modify the profile attributes of the user. If the user has administrative privileges, the application could do much more damage.

[Figure B–1](#) illustrates a typical OpenSSO Enterprise deployment within an enterprise. While the figure helps to define security issues related to cookie hijacking, it also helps to define the solution. Therefore, the figure applies to a deployment where the tasks presented subsequently in this document have already been performed.

The deployment illustrated in the figure uses SSO. Moreover, as part of the solution, the OpenSSO Enterprise implementation of CDSSO is enabled. To guard against potential threats posed by cookie hijacking, CDSSO enablement is required regardless of the number of domains involved. Having CDSSO enabled when the deployment involves a single domain might seem counterintuitive, but ultimately security is increased by taking advantage of the CDSSO framework. For other information about the use of CDSSO in this type of deployment, see [“Enabling OpenSSO Enterprise to Use Unique SSO Tokens” on page 108](#).

The numbers in the figure correspond to the numbered explanations that follow. The explanations combine to provide an outline of the process that takes place when a request is made for a protected resource, emphasizing how the SSO token flows between components. The deployment includes a single virtual AuthN (Authentication) and AuthZ (Policy) server (denoted as OpenSSO Enterprise or Identity Provider in the figure), and a number of applications (Service Providers), denoted as Trusted Application and Untrusted Application in the diagram. The Service Providers are usually front-ended with an agent (specifically, an agent from the Policy Agent 3.0 software set) that handles the SSO (AuthN) and Policy (AuthZ).

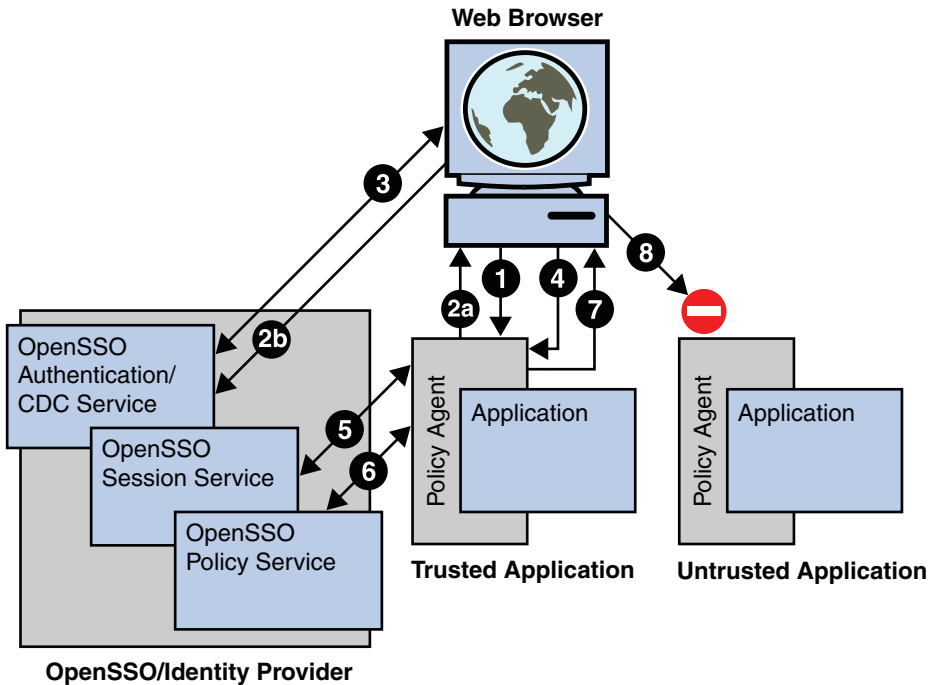


FIGURE B-1 The Role of the Session Cookie in Allowing Access to Protected Web Resources

Note – Figure B-1 does not include every detail involved in the flow of the SSO token. The figure provides a simplified view that corresponds to the scope of this document.

1. The user accesses an application through a web browser. The agent, which is an agent from the Policy Agent 3.0 software set, intercepts the request and checks for the user's privilege to access the application. The check is specifically a check for a valid OpenSSO Enterprise SSO token, which is set as a session cookie in the user's browser.
2. Assuming the user does not have a valid SSO token, the agent redirects the browser to OpenSSO Enterprise Authentication Service. The agent also provides its identity using the Liberty AuthN request specification. Since this single redirection is a two step process, it is illustrated in the figure as two substeps: 2A and 2B.
3. OpenSSO Enterprise Authentication Service authenticates the user and, after successful authentication, redirects the user back to the target application with the SSO token as part of the URL query parameter (in a format specified by Liberty AuthN response specification).
4. The agent receives a successful AuthN response from OpenSSO Enterprise Authentication Service. It gets the SSO token and sets it as a session cookie for the host (rather than for the domain) to the browser's request.

5. The agent validates the SSO token with OpenSSO Enterprise Session Service.
6. After a successful SSO token validation in Step 5, the agent then checks the permissions for the user to access the application with OpenSSO Enterprise Policy Service.
7. If permission is granted in Step 6, the user is allowed access to the application.
8. As indicated in the figure with an incomplete connection to Untrusted Application, the same SSO token cannot be used to gain access to an untrusted application. OpenSSO Enterprise denies such access since the SSO token is unique to the application and may not be shared or reissued to other agents or applications.

OpenSSO Enterprise Solution: Cookie Hijacking Security Issues

This section explains how performing the tasks described in this document enables OpenSSO Enterprise to handle the security issues discussed in the preceding section, [“Defining Key Cookie Hijacking Security Issues” on page 101](#).

Note – When applications use a secure protocol such as HTTPS, the SSO Token is not visible to network snooping. This security issue is labeled “Security Issue: Insecure Protocol” in this document. Ensuring that all protected resources use a secure protocol is not a security measure administered using OpenSSO Enterprise, but this a very prudent security measure that you should consider implementing if it is not currently in place.

OpenSSO Enterprise Solution: Shared Session Cookies

The security issue labeled “Security Issue: Shared Session Cookies” in this document pertains to applications sharing the same HTTP or HTTPS session cookie. OpenSSO Enterprise addresses this security threat by issuing a unique SSO token to each Application/Agent after the user has been authenticated. The unique SSO token is referred to as a “restricted token.”

The term “Application/Agent,” indicates that the restricted token is inextricably connected to the application and to the agent (which specifically refers to an agent from the Policy Agent 3.0 software set). Since each user's SSO token is unique for each Application/Agent, the increased security provided by this scenario prevents an untrusted application, impersonating the user, from accessing other applications. More specifically, since the SSO token (restricted token) assigned to a user (as a part of the user's session) is associated with the agent that did the initial redirection for authentication, all subsequent requests are checked to verify that they are coming from the same agent. Thus, if a hacker tries to use the same restricted token to access another application, a security violation is thrown.

What makes the restricted token “restricted” is not related to the syntax of the token. The syntax of a restricted token is the same as that of a regular SSO token. Instead, a specific constraint is associated with the restricted token. This constraint is what ensures that the restricted token is only used for an application that a given agent protects.

OpenSSO Enterprise Solution: A Less Secure Application

The security issue labeled “Security Issue: A Less Secure Application” in this document pertains to the potential threat of applications that are “less secure.” With the OpenSSO Enterprise solution, if one application is somehow compromised, the hacker cannot hack into other applications.

OpenSSO Enterprise Solution: Modification of Profile Attributes

The security issue labeled “Security Issue: Access to User Profile Attributes” in this document pertains to the threat posed by an untrusted application modifying the profile attributes of the user. The OpenSSO Enterprise solution to this issue does not change the SSO token. The restricted SSO token is identical to the regular SSO token ID. However, the set of Session Service operations that accept restricted SSO token IDs is limited. This functionality enables OpenSSO Enterprise to prevent applications from modifying profile attributes of the user.

Key Aspects of the OpenSSO Enterprise Solution: Cookie Hijacking Security Issues

The following subsections explain some of the key or more complex aspects of the OpenSSO Enterprise solution to the cookie hijacking security issues defined in this document.

The Significance of Creating an Agent Profile for Each Agent

A key task presented in this document is the task of ensuring that each agent has its own agent profile. For Policy Agent 3.0, J2EE agents and web agents differ in regard to the agent profile. Each J2EE agent must have a unique agent profile to function. Web agents, on the other hand, have a default value for the agent profile that all web agent instances can share. Therefore, to configure web agents against session-cookie hijacking, you must perform additional configuration steps after a web agent is installed to ensure that each web agent has a unique agent profile.

Basically, all the configuration steps presented in this document rely on each agent having its own agent profile. As explained in [“OpenSSO Enterprise Solution: Shared Session Cookies” on page 105](#), a situation where applications share session cookies presents a security issue. OpenSSO Enterprise addresses this potential security risk by issuing a unique SSO token to each agent. In order to issue a unique SSO token to each agent, each agent must have its own agent profile.

OpenSSO Enterprise Session Cookies Involved in Issuing Unique SSO Tokens

When OpenSSO Enterprise is configured to issue unique SSO tokens for each Application/Agent, the following cookies are involved:

Cookie Name	Cookie Value (place holder)	Example Cookie Domain Information
iPlanetDirectoryPro	SSO-token	amHost.example.com

The value of this cookie, which is represented in the preceding table with the place holder *SSO-token*, is the actual value of the token. The domain is set to the host name of the OpenSSO Enterprise instance where the user was authenticated.

Cookie Name	Cookie Value (place holder)	Example Cookie Domain Information
iPlanetDirectoryPro	restricted-SSO-token	agentHost.example.com

The value of this cookie, which is represented in the preceding table with the place holder *restricted-SSO-token*, is the actual value of the token. The domain is set to the host name of the agent instance for which the restricted token is issued.

Cookie Name	Example Cookie Value	Example Cookie Domain Information
sunIdentityServerAuthNServer	https://amHost.example.com:8080	.example.com

The value of this cookie, which is represented in the preceding table with the example URL `https://amHost.example.com:8080`, is the URL of the OpenSSO Enterprise instance where the user was authenticated. The protocol used for this particular example is HTTPS while the port number is a non-default example, 8080. The domain must be set such that it covers all the instances of OpenSSO Enterprise installed on the network.

Enabling OpenSSO Enterprise to Use Unique SSO Tokens

To enable OpenSSO Enterprise to issue unique SSO tokens, you must enable CDSSO. Therefore, though CDSSO is usually enabled for multiple-domain deployments, in this case, CDSSO must be enabled whether the entire deployment is on a single domain or is spread across multiple domains. In no way does enabling CDSSO for a single domain negatively affect the deployment.

The next section describes the steps required to configure OpenSSO Enterprise to prevent session-cookie hijacking from causing a breach of security.

Implementing the OpenSSO Enterprise Solution for Cookie Hijacking Security Issues

The instructions presented in this section provide a solution to the potential risks related to session-cookie hijacking as outlined in this document. This section includes two subsections as follows:

- [“Creating or Updating an Agent Profile” on page 108](#)
- [“Configuring the OpenSSO Enterprise Deployment Against Cookie Hijacking” on page 111](#)

In Policy Agent 3.0, web agents and J2EE agents use agent profiles in a slightly different way. The information provided in this section on the agent profile explains the differences between the two agent types and provides the actual steps necessary to ensure that each agent has its own agent profile.

This section also provides the other configuration steps necessary to guard web resources in an OpenSSO Enterprise deployment against the threat of session-cookie hijacking.

After you perform the tasks presented in this document, OpenSSO Enterprise starts enforcing restrictions on the sessions it creates. The new configuration enables OpenSSO Enterprise to more closely track aspects of each session. At that point, not only does OpenSSO Enterprise record which agent performed the initial redirection for authentication it also tracks the applications to which an SSO token has been issued. OpenSSO Enterprise uses this information to facilitate the processing of each subsequent request and to prevent unauthorized access to protected web resources.

Creating or Updating an Agent Profile

Since J2EE agents and web agents in the Policy Agent 3.0 software set handle the agent profile differently, the configuration steps can differ. The following table provides information about the agent profile in relationship to the two agent types:

Agent Profile Information	Web Agents	J2EE Agents
The agent profile includes a user ID and password that the agent uses to log in to OpenSSO Enterprise in order to request services.	Yes	Yes
The agent has default credentials for the agent profile.	Yes	No

As the preceding table indicates, J2EE agents in the 3.0 release do not provide default credentials for the agent profile. You create an agent profile in OpenSSO Enterprise Console prior to installing the J2EE agent, which allows you to provide information about the agent profile during installation. If you do not create an agent profile, the J2EE agent will not function. Since each successfully installed J2EE agent has a unique agent profile, no further configuration would be required regarding the agent profile credentials.



Caution – This document provides guidance for creating or updating agent profiles. However, the steps can vary depending upon the agent type and the status of the agent. While the steps that follow are appropriate for many scenarios, such as for agents that are yet to be installed, the steps are not appropriate for all possible scenarios.

For example, if you are using a J2EE agent that was previously installed, the pre-existing agent profile is appropriate. You would not be required to perform any of the steps presented in this section about creating or updating the agent profile.

However, in terms of a previously installed J2EE agent, if you decide to change the agent profile user ID and password, even though not required, then configuration steps are necessary. Be aware that the steps required in this scenario are not provided in this document. For such a scenario, refer to the Policy Agent 3.0 documentation for information specifically about *updating* the agent profile.

For more information on creating or updating the agent profile, see the documentation for the specific agent you are using.

▼ To Create an Agent Profile

This task is appropriate for web agents and J2EE agents. See the preceding Caution for information about when this task might apply.

Agent profiles apply to specific agents but are created and modified in OpenSSO Enterprise Console. Therefore, tasks related to the agent profile are discussed in OpenSSO Enterprise documentation and Policy Agent documentation, as well as in this document.

Perform the following steps using the OpenSSO Enterprise Console. The key steps of this task involve creating an agent ID and an agent password.

- 1 **With the Access Control tab selected click the name of the realm for which you would like to create an agent profile.**
- 2 **Select the Subjects tab.**
- 3 **Select the Agent tab.**
- 4 **Click New.**
- 5 **Enter values for the following fields:**
 - ID.** Enter the name or identity of the agent. This is the agent profile name, which is the name the agent uses to log into Access Manager. Multi-byte names are not accepted.
 - Password.** Enter the agent password. This password must be different than the LDAP user password used by OpenSSO Enterprise.
 - Password (confirm).** Confirm the password.
 - Device Status.** Select the device status of the agent. The default status is Active. If set to Active, the agent will be able to authenticate to and communicate with Access Manager. If set to Inactive, the agent will not be able to authenticate to Access Manager.
- 6 **Click Create.**

The list of agents appears.
- 7 **(Optional) If you desire, add a description to your newly created agent profile:**
 - a. **Click the name of your newly created agent profile from the agent list.**
 - b. **In the Description field, enter a brief description of the agent.**

For example, you can enter the agent instance name or the name of the application it is protecting.
 - c. **Click Save.**

Next Steps If you performed this task for a web agent, you must also perform the following task about updating the agent profile. If you performed this task for a J2EE agent, you can skip to [“Configuring the OpenSSO Enterprise Deployment Against Cookie Hijacking” on page 111.](#)

Configuring the OpenSSO Enterprise Deployment Against Cookie Hijacking

At this point in the configuration, each agent has its own agent profile. However, OpenSSO Enterprise has not been configured yet to associate an SSO token to a specific agent profile. The steps in this section enable this type of association. Ultimately, the new configuration introduces “restricted tokens” into the OpenSSO Enterprise deployment, guarding against security issues as described in this document.

▼ To Configure the OpenSSO Enterprise Deployment Against Cookie Hijacking

This task description includes configuration information for agents in the Policy Agent 3.0 software set. Perform the task on every agent instance for which you want to enhance security. The best practice is to perform the task on all the agent instances in the OpenSSO Enterprise deployment. As part of the configuration of each agent instance, you must also make specific configurations directly to OpenSSO Enterprise. For this task, be prepared to access the OpenSSO Enterprise Console, the OpenSSO Enterprise `AMConfig.properties` configuration file, and a browser that can access a protected web resource.

1 Using the OpenSSO Enterprise Administration Console, access the agent profile configuration page.

For the steps on navigating within the OpenSSO Enterprise Administration Console to the agent profile configuration page, see [“To Create an Agent Profile” on page 109](#).

2 Add the appropriate value to the field labeled Agent Key Value.

Set the agent properties with a key/value pair as illustrated in the example that follows. This property is used by OpenSSO Enterprise to retrieve an agent profile from an agent repository for credential assertions about agents. Currently, only one property is valid. All other properties are ignored. Use the following format:

```
agentRootURL=protocol://hostname:port/
```

The preceding entry must be precise. Be aware that the string “agentRootURL” is case sensitive. Also, the slash following the port number is required.

protocol Represents the protocol used, such as HTTP or HTTPS.

hostname Represents the host name of the machine on which the agent resides. This machine also hosts the resources that the agent protects.

port Represents the port number on which the agent is installed. The agent listens to incoming traffic on this port and, from the port, intercepts all requests to access resources on the host.

The following is an example of how this property could be set:

```
agentRootURL=https://agentHost.example.com:8080/
```

3 Edit the appropriate agent properties as described in the substeps that follow:

a. Set the property that enables CDSSO to true as illustrated:

- For J2EE agents set the following property as indicated:

```
com.sun.identity.agents.config.cdsso.enable = true
```

- For web agents set the following property as indicated:

```
com.sun.am.policy.agents.config.cdsso.enable = true
```

The preceding property setting enables CDSSO, which is required for each agent instance since the agent will use functionality provided by the CDSSO feature.

b. Set the property that stores the URL users are directed to after they log in successfully in a deployment enabled for CDSSO:

- For J2EE agents set the corresponding property as suggested by the following example:

```
com.sun.identity.agents.config.cdsso.cdcservlet.url[0] =  
https://amHost.example.com:8080/amserver/cdcservlet
```

- For web agents set the corresponding property as suggested by the following example:

```
com.sun.am.policy.agents.config.cdcservlet.url =  
https://amHost.example.com:8080/amserver/cdcservlet
```

4 Restart the container that hosts the agent.

5 Edit the OpenSSO Enterprise AMConfig.properties configuration file to reflect the required changes.

a. Set the following property to true as illustrated:

```
com.sun.identity.enableUniqueSSOTokenCookie = true
```

b. Set the following property exactly as it is illustrated:

```
com.sun.identity.authentication.uniqueCookieName =  
sunIdentityServerAuthNServer
```


- c. Set the following property to a domain such that it covers all the OpenSSO Enterprise instances installed:

```
com.sun.identity.authentication.uniqueCookieDomain
```

The following example illustrates how this property would be set if the domain name was `example.com`.

```
com.sun.identity.authentication.uniqueCookieDomain = .example.com
```

- 6 In the OpenSSO Enterprise Administration Console, select the Configuration tab.

- 7 Scroll as needed to the System Properties list and click Platform.

- 8 In the Cookie Domain list, change the cookie domain name.

This step enables OpenSSO Enterprise to set host-specific session cookies instead of domain-wide session cookies.

- a. Ensure that the default domain, such as “`example.com`,” is selected.

- b. Click Remove.

- c. Enter the name of the machine hosting the OpenSSO Enterprise instance.

For example:

`amHost.example.com`

- d. Click Add.

- 9 Ensure that the proper cookies appear in a browser.

- a. Use a browser to access a resource that is protected by the agent that you just configured.

- b. Check the browser's cookie settings to ensure that the three following cookies appear:

Cookie Name	Example Cookie Value	Example Cookie Domain Information
iPlanetDirectoryPro	<i>SSO-token</i>	<code>amHost.example.com</code>
iPlanetDirectoryPro	<i>restricted-SSO-token</i>	<code>agentHost.example.com</code>
sunIdentityServerAuthNServer	<code>https://amHost.example.com:8080</code>	<code>.example.com</code>

For more information about the preceding cookies, see “[OpenSSO Enterprise Session Cookies Involved in Issuing Unique SSO Tokens](#)” on page 107.

Using the ssoadm Command-Line Utility With Agents

When the agent configuration is centralized, you can configure OpenSSO Enterprise through the 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 the OpenSSO Enterprise Console can also be made using the command line. This appendix indicates which ssoadm subcommands are related to agents.

An ssoadm Command-Line Example Specific to Agents

This section provides an example of how you can use the ssoadm command-line for agent-related subcommands. This example highlights the update-agent option. The update-agent option allows you to configure agent properties. The following is an example of how the ssoadm command can be issued with the update-agent option.

```
# ./ssoadm update-agent -e testRealm1 -b testAgent1 -u amadmin -f
/tmp/testpwd -a "com.sun.identity.agents.config.notenforced.uri[0]=/exampleDir/public/*"
```

Remark C-1 **Reviewer** Please look at the command above. Are all the details of it accurate. It seems that the above example is different for web agents and J2EE agents. Besides that example above, does this appendix apply equally to web agents and J2EE agents? Are there any differences that should be addressed?

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 A, “Wildcard Matching in Policy Agent 3.0 J2EE Agents.”](#)



Caution – When issuing the `ssoadm` command, if you include values that contain wildcards (* or -*), then the property name/value pair should be enclosed in double quotes to avoid substitution by the shell. This applies when you use the `-a (--attributevalues)` option. The double quotes are not necessary when you list the properties in a data file and access them with the `-D` option.

The format used to assign values to agent properties differs for the OpenSSO Enterprise Console and the `ssoadm` command-line utility. For the `ssoadm` utility, refer to the agent property files: `OpenSSOAgentBootstrap.properties` and `OpenSSOAgentConfiguration.properties`. These files demonstrate the correct format to use when assigning values to the agent properties using the `ssoadm` utility. Find these property files on the agent host machine in the following directory:

PolicyAgent-base/AgentInstance-Dir/config

Listing the Options for an ssoadm Subcommand

You can read the options for a subcommand from this guide or you can list the options yourself while using the command. On the machine hosting OpenSSO Enterprise, in the directory containing the `ssoadm` utility, issue the `ssoadm` command with the appropriate subcommand. For example:

```
# ./ssoadm update-agent
```

Since the preceding command is missing required options, the utility merely lists all the options available for this subcommand. For example:

```
ssoadm update-agent --options [--global-options]
Update agent configuration.
Usage:
ssoadm
    --realm|-e
    --agentname|-b
    --adminid|-u
    --password-file|-f
    [--set|-s]
    [--attributevalues|-a]
    [--datafile|-D]Global Options:
    --locale, -l
        Name of the locale to display the results.
```

```
--debug, -d
    Run in debug mode. Results sent to the debug file.

--verbose, -v
    Run in verbose mode. Results sent to standard output.
```

Options:

```
--realm, -e
    Name of realm.

--agentname, -b
    Name of agent.

--adminid, -u
    Administrator ID of running the command.

--password-file, -f
    File name that contains password of administrator.

--set, -s
    Set this flag to overwrite properties values.

--attributevalues, -a
    properties e.g. homeaddress=here.

--datafile, -D
    Name of file that contains properties.
```

Analysis of an ssoadm Subcommand's Usage Information

By looking at the usage information of a subcommand, you can determine which options are required and which are optional. You can list an option for the command with either a single letter, such as `-e` or with an entire word, such as `--realm`. The following is a list of the usage information for the `update-agent` subcommand:

```
ssoadm update-agent
  --realm|-e
  --agentname|-b
  --adminid|-u
  --password-file|-f
  [--set|-s]
  [--attributevalues|-a]
  [--datafile|-D]
```

The options not bounded by square brackets are required. Therefore, `realm`, `agentname`, `adminid`, `password-file`. However, even though the three options in brackets (the global options) are considered optional, you must use either `--attributevalues` or `--datafile` to

provide a property name and the corresponding value. The `--attributevalues` option is appropriate for assigning values to a single property. The `--datafile` option is appropriate for setting several properties at once. The `realm` and `agentname` options identify the specific agent you are configuring. The `adminid` and `password-file` commands identify you as someone who has the right to configure this agent.

The following command serves as an example of how you can change several agent properties at once. In this scenario the properties and their respective values are stored in a file, `/tmp/testproperties`, to which the command points:

```
# ./ssoadm update-agent -e testRealm1 -b testAgent1 -u amadmin -f
/tmp/testpwd -D /tmp/testproperties
```

Agent-Related Subcommands for the ssoadm Command

This sections lists the options available for each of the agent-related subcommands of the `ssoadm` command. The agent-related subcommands are presented as links in the following list:

- [“The ssoadm Command: add-agent-to-grp subcommand” on page 118](#)
- [“The ssoadm Command: agent-remove-props subcommand” on page 119](#)
- [“The ssoadm Command: create-agent subcommand” on page 120](#)
- [“The ssoadm Command: create-agent-grp subcommand” on page 121](#)
- [“The ssoadm Command: delete-agent-grps subcommand” on page 122](#)
- [“The ssoadm Command: delete-agents subcommand” on page 123](#)
- [“The ssoadm Command: list-agent-grp-members subcommand” on page 124](#)
- [“The ssoadm Command: list-agent-grps subcommand” on page 125](#)
- [“The ssoadm Command: list-agents subcommand” on page 125](#)
- [“The ssoadm Command: remove-agent-from-grp subcommand” on page 126](#)
- [“The ssoadm Command: show-agent subcommand” on page 127](#)
- [“The ssoadm Command: show-agent-grp subcommand” on page 128](#)
- [“The ssoadm Command: show-agent-membership subcommand” on page 129](#)
- [“The ssoadm Command: show-agent-types subcommand” on page 130](#)
- [“The ssoadm Command: update-agent subcommand” on page 130](#)
- [“The ssoadm Command: update-agent-grp subcommand” on page 131](#)

The ssoadm Command: add-agent-to-grp subcommand

```
ssoadm add-agent-to-grp --options [--global-options]
```

Add agents to a agent group.

Usage:

ssoadm

```
--realm|-e
--agentgroupname|-b
--agentnames|-s
--adminid|-u
--password-file|-f
```

Global Options:

```
--locale, -l
    Name of the locale to display the results.

--debug, -d
    Run in debug mode. Results sent to the debug file.

--verbose, -v
    Run in verbose mode. Results sent to standard output.
```

Options:

```
--realm, -e
    Name of realm.

--agentgroupname, -b
    Name of agent group.

--agentnames, -s
    Names of agents.

--adminid, -u
    Administrator ID of running the command.

--password-file, -f
    File name that contains password of administrator.
```

The ssoadm Command: agent-remove-props subcommand

```
ssoadm agent-remove-props --options [--global-options]
Remove agent's properties.
```

Usage:

```
ssoadm
    --realm|-e
    --agentname|-b
    --attributenames|-a
    --adminid|-u
    --password-file|-f
```

Global Options:

```
--locale, -l
    Name of the locale to display the results.

--debug, -d
    Run in debug mode. Results sent to the debug file.

--verbose, -v
    Run in verbose mode. Results sent to standard output.
```

Options:

```
--realm, -e
    Name of realm.

--agentname, -b
    Name of agent.

--attributenames, -a
    properties name(s).

--adminid, -u
    Administrator ID of running the command.

--password-file, -f
    File name that contains password of administrator.
```

The ssoadm Command: create-agent subcommand

```
ssoadm create-agent --options [--global-options]
```

Create a new agent configuration.

Usage:

ssoadm

```
--realm|-e
--agentname|-b
--agenttype|-t
--adminid|-u
--password-file|-f
[--attributevalues|-a]
[--datafile|-D]
```

Global Options:

```
--locale, -l
    Name of the locale to display the results.

--debug, -d
    Run in debug mode. Results sent to the debug file.
```


--verbose, -v
Run in verbose mode. Results sent to standard output.

Options:

--realm, -e
Name of realm.

--agentname, -b
Name of agent.

--agenttype, -t
Type of agent. e.g. J2EEAgent, WebAgent

--adminid, -u
Administrator ID of running the command.

--password-file, -f
File name that contains password of administrator.

--attributevalues, -a
properties e.g. homeaddress=here.

--datafile, -D
Name of file that contains properties.

The ssoadm Command: create-agent-grp subcommand

ssoadm create-agent-grp --options [--global-options]
Create a new agent group.

Usage:

ssoadm

--realm|-e
--agentgroupname|-b
--agenttype|-t
--adminid|-u
--password-file|-f
[--attributevalues|-a]
[--datafile|-D]

Global Options:

--locale, -l
Name of the locale to display the results.

--debug, -d

Run in debug mode. Results sent to the debug file.

--verbose, -v

Run in verbose mode. Results sent to standard output.

Options:

--realm, -e

Name of realm.

--agentgroupname, -b

Name of agent group.

--agenttype, -t

Type of agent group. e.g. J2EEAgent, WebAgent

--adminid, -u

Administrator ID of running the command.

--password-file, -f

File name that contains password of administrator.

--attributevalues, -a

properties e.g. homeaddress=here.

--datafile, -D

Name of file that contains properties.

The ssoadm Command: delete-agent-grps subcommand

ssoadm delete-agent-grps --options [--global-options]

Delete agent groups.

Usage:

ssoadm

--realm|-e

--agentgroupnames|-s

--adminid|-u

--password-file|-f

Global Options:

--locale, -l

Name of the locale to display the results.

--debug, -d

Run in debug mode. Results sent to the debug file.

```
--verbose, -v  
    Run in verbose mode. Results sent to standard output.
```

Options:

```
--realm, -e  
    Name of realm.  
  
--agentgroupnames, -s  
    Names of agent group.  
  
--adminid, -u  
    Administrator ID of running the command.  
  
--password-file, -f  
    File name that contains password of administrator.
```

The ssoadm Command: delete-agents subcommand

```
ssoadm delete-agents --options [--global-options]  
Delete agent configurations.
```

Usage:

```
ssoadm
```

```
--realm|-e  
--agentnames|-s  
--adminid|-u  
--password-file|-f
```

Global Options:

```
--locale, -l  
    Name of the locale to display the results.  
  
--debug, -d  
    Run in debug mode. Results sent to the debug file.  
  
--verbose, -v  
    Run in verbose mode. Results sent to standard output.
```

Options:

```
--realm, -e  
    Name of realm.  
  
--agentnames, -s  
    Names of agent.  
  
--adminid, -u
```

Administrator ID of running the command.

`--password-file, -f`

File name that contains password of administrator.

The ssoadm Command: list-agent-grp-members subcommand

`ssoadm list-agent-grp-members --options [--global-options]`

List agents in agent group.

Usage:

`ssoadm`

`--realm|-e`

`--agentgroupname|-b`

`--adminid|-u`

`--password-file|-f`

`[--filter|-x]`

Global Options:

`--locale, -l`

Name of the locale to display the results.

`--debug, -d`

Run in debug mode. Results sent to the debug file.

`--verbose, -v`

Run in verbose mode. Results sent to standard output.

Options:

`--realm, -e`

Name of realm.

`--agentgroupname, -b`

Name of agent group.

`--adminid, -u`

Administrator ID of running the command.

`--password-file, -f`

File name that contains password of administrator.

`--filter, -x`

Filter (Pattern).

The ssoadm Command: list-agent-grps subcommand

ssoadm list-agent-grps --options [--global-options]
List agent groups.

Usage:

```
ssoadm
  --realm|-e
  --adminid|-u
  --password-file|-f
  [--filter|-x]
  [--agenttype|-t]
```

Global Options:

```
--locale, -l
    Name of the locale to display the results.

--debug, -d
    Run in debug mode. Results sent to the debug file.

--verbose, -v
    Run in verbose mode. Results sent to standard output.
```

Options:

```
--realm, -e
    Name of realm.

--adminid, -u
    Administrator ID of running the command.

--password-file, -f
    File name that contains password of administrator.

--filter, -x
    Filter (Pattern).

--agenttype, -t
    Type of agent. e.g. J2EEAgent, WebAgent
```

The ssoadm Command: list-agents subcommand

ssoadm list-agents --options [--global-options]
List agent configurations.

Usage:

```
ssoadm
  --realm|-e
```

```
--adminid|-u
--password-file|-f
[--filter|-x]
[--agenttype|-t]
```

Global Options:

```
--locale, -l
    Name of the locale to display the results.

--debug, -d
    Run in debug mode. Results sent to the debug file.

--verbose, -v
    Run in verbose mode. Results sent to standard output.
```

Options:

```
--realm, -e
    Name of realm.

--adminid, -u
    Administrator ID of running the command.

--password-file, -f
    File name that contains password of administrator.

--filter, -x
    Filter (Pattern).

--agenttype, -t
    Type of agent. e.g. J2EEAgent, WebAgent
```

The ssoadm Command: remove-agent-from-grp subcommand

ssoadm remove-agent-from-grp --options [--global-options]
Remove agents from a agent group.

Usage:

```
ssoadm --realm|-e
    --agentgroupname|-b
    --agentnames|-s
    --adminid|-u
    --password-file|-f
```

Global Options:

```
--locale, -l
```

Name of the locale to display the results.

--debug, -d

Run in debug mode. Results sent to the debug file.

--verbose, -v

Run in verbose mode. Results sent to standard output.

Options:

--realm, -e

Name of realm.

--agentgroupname, -b

Name of agent group.

--agentnames, -s

Names of agents.

--adminid, -u

Administrator ID of running the command.

--password-file, -f

File name that contains password of administrator.

The ssoadm Command: show-agent subcommand

ssoadm show-agent --options [--global-options]

Show agent profile.

Usage:

ssoadm

--realm|-e

--agentname|-b

--adminid|-u

--password-file|-f

[--outfile|-o]

[--inherit|-i]

Global Options:

--locale, -l

Name of the locale to display the results.

--debug, -d

Run in debug mode. Results sent to the debug file.

--verbose, -v

Run in verbose mode. Results sent to standard output.

Options:

- realm, -e
Name of realm.
- agentname, -b
Name of agent.
- adminid, -u
Administrator ID of running the command.
- password-file, -f
File name that contains password of administrator.
- outfile, -o
Filename where configuration is written to.
- inherit, -i
Set this to inherit properties from parent group.

The ssoadm Command: show-agent-grp subcommand

ssoadm show-agent-grp --options [--global-options]
Show agent group profile.

Usage:

ssoadm

- realm|-e
- agentgroupname|-b
- adminid|-u
- password-file|-f
- [--outfile|-o]

Global Options:

- locale, -l
Name of the locale to display the results.
- debug, -d
Run in debug mode. Results sent to the debug file.
- verbose, -v
Run in verbose mode. Results sent to standard output.

Options:

- realm, -e
Name of realm.


```
--agentgroupname, -b
    Name of agent group.

--adminid, -u
    Administrator ID of running the command.

--password-file, -f
    File name that contains password of administrator.

--outfile, -o
    Filename where configuration is written to.
```

The ssoadm Command: show-agent-membership subcommand

```
ssoadm show-agent-membership --options [--global-options]
List agent?s membership.
```

Usage:

```
ssoadm
    --realm|-e
    --agentname|-b
    --adminid|-u
    --password-file|-f
```

Global Options:

```
--locale, -l
    Name of the locale to display the results.

--debug, -d
    Run in debug mode. Results sent to the debug file.

--verbose, -v
    Run in verbose mode. Results sent to standard output.
```

Options:

```
--realm, -e
    Name of realm.

--agentname, -b
    Name of agent.

--adminid, -u
    Administrator ID of running the command.

--password-file, -f
    File name that contains password of administrator.
```

The ssoadm Command: show-agent-types subcommand

ssoadm show-agent-types --options [--global-options]

Show agent types.

Usage:

ssoadm

--adminid|-u
--password-file|-f

Global Options:

--locale, -l
Name of the locale to display the results.

--debug, -d
Run in debug mode. Results sent to the debug file.

--verbose, -v
Run in verbose mode. Results sent to standard output.

Options:

--adminid, -u
Administrator ID of running the command.

--password-file, -f
File name that contains password of administrator.

The ssoadm Command: update-agent subcommand

ssoadm update-agent --options [--global-options]

Update agent configuration.

Usage:

ssoadm

--realm|-e
--agentname|-b
--adminid|-u
--password-file|-f
[--set|-s]
[--attributevalues|-a]
[--datafile|-D]

Global Options:

--locale, -l
Name of the locale to display the results.

--debug, -d
Run in debug mode. Results sent to the debug file.

--verbose, -v
Run in verbose mode. Results sent to standard output.

Options:

--realm, -e
Name of realm.

--agentname, -b
Name of agent.

--adminid, -u
Administrator ID of running the command.

--password-file, -f
File name that contains password of administrator.

--set, -s
Set this flag to overwrite properties values.

--attributevalues, -a
properties e.g. homeaddress=here.

--datafile, -D
Name of file that contains properties.

The ssoadm Command: update-agent-grp subcommand

ssoadm update-agent-grp --options [--global-options]

Update agent group configuration.

Usage:

```
ssoadm
  --realm|-e
  --agentgroupname|-b
  --adminid|-u
  --password-file|-f
  [--set|-s]
  [--attributevalues|-a]
  [--datafile|-D]
```

Global Options:

--locale, -l

Name of the locale to display the results.

--debug, -d

Run in debug mode. Results sent to the debug file.

--verbose, -v

Run in verbose mode. Results sent to standard output.

Options:

--realm, -e

Name of realm.

--agentgroupname, -b

Name of agent group.

--adminid, -u

Administrator ID of running the command.

--password-file, -f

File name that contains password of administrator.

--set, -s

Set this flag to overwrite properties values.

--attributevalues, -a

properties e.g. homeaddress=here.

--datafile, -D

Name of file that contains properties.

Index

A

- agent profile
 - and agentadmin --encrypt, 54-55
 - creating, 64-66
- agentadmin command, 47-59
 - agentInfo, 52
 - encrypt, 53-55
 - getEncryptKey, 55-56
 - getUuid, 57-58
 - help, 59
 - install, 48-49
 - listAgents, 51
 - uninstall, 49-51
 - uninstallAll, 56
 - usage, 58-59
 - version, 52-53
- agentadmin program, 47-59
- authentication
 - level, 23
 - user, 22, 27, 33, 35

C

- Class AmFilterManager, 94-95
- Class AmSSOCache, 95-96
- compatibility, OpenSSO Enterprise, J2EE agents, 17
- creating, agent profile, 64-66
- cross-domain single sign-on, 20

F

- form login, customizing the agent response, 79-81

I

- Interface IAmSSOCache, 95
- inverting
 - not-enforced
 - URI list, 85

J

- J2EE
 - security, 35, 38

L

- LDAP
 - attributes
 - as cookies, 89
 - as HTTP headers, 88-89
 - as request attributes, 89
- logging, access attempt, 20

N

- not-enforced
 - URI list, 84-85
 - inverting, 85

O

- OpenSSO Enterprise
 - agent profile
 - and agentadmin --encrypt, 54-55
 - creating, 64-66
 - Service
 - Authentication, 20, 22, 27
 - Logging, 20
 - Naming, 20
 - Policy, 20, 28
 - Session, 20, 35
 - version 6.3
 - configuration, 64

- user

- authentication, 22, 27, 33, 35

W

- web server, embedded, 38
- web tier declarative security, 76-81

P

- password file, and agentadmin --encrypt, 53-55
- policy decision
 - enforcement, 22, 37
 - process, 23
- portal server, 34, 37

R

- redirect
 - browser, 27, 35

S

- security
 - J2EE, 35, 38
- service, OpenSSO Enterprise, 20
- session
 - token, 28
 - web server agent, 34
- single sign-on, enforcement, 37

U

- URI, 37