Sun Java System Access Manager Policy Agent 2.2 Guide for Apache Tomcat 6.0

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

Part No: 820–4802–11
March 13, 2009

090313@21990

Contents

Preface

This Sun Java System Access Manager Policy Agent 2.2 Guide for Apache Tomcat 6.0 is a Java 2 Platform Enterprise Edition (J2EE) agent guide. Therefore, it provides general information about J2EE agents in the Sun Java™ System Access Manager Policy Agent 2.2 software set. This guide also provides specific information about Sun Java System Access Manager Policy Agent for Apache Tomcat 6.0.

For more support and compatibility information specific to this J2EE agent, see "Supported Platforms and Compatibility of Agent for Apache Tomcat 6.0" on page 56.

Included in this guide is information about installing, configuring, uninstalling, and troubleshooting J2EE agents, again with the focus being on Agent for Apache Tomcat 6.0.

Who Should Use This Book

This *Sun Java System Access Manager Policy Agent 2.2 Guide for Apache Tomcat 6.0* is intended for use by IT professionals who manage access to their network using Sun Java System servers and software. 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
- Enterprise Java Beans (EJB)

Before You Read This Book

Sun Java System Policy Agent software works with Sun Java System Access Manager. Both products work with Sun Java Enterprise System, a software infrastructure that supports enterprise applications distributed across a network or Internet environment. Furthermore, Sun Java System Directory Server is a necessary component in a new Access Manager deployment since it is used as the data store. To understand how these products interact and to understand this book, you should be familiar with the following documentation:

- Sun Java Enterprise System documentation set, which can be accessed online at `http://docs.sun.com`. All Sun technical documentation is available online through this web site, including the other documentation sets referred to in this list.

  You can browse the documentation archive or search for a specific book title, part number, or subject.
- Sun Java System Directory Server documentation set.
- Sun Java System Access Manager documentation set, which is explained in more detail subsequently in this chapter.
- Sun Java System Access Manager Policy Agent 2.2 documentation set, which is explained in more detail subsequently in this chapter.

How This Book Is Organized

This book is organized in the following manner:

Preface, this chapter, provides information about this book to help you use the book to your best advantage.

Chapter 1, "Introduction to J2EE Agents for Policy Agent 2.2," introduces J2EE agents in Policy Agent 2.2, focusing on what all J2EE agents have in common in this release.

Chapter 2, "Vital Installation Information for a J2EE Agent in Policy Agent 2.2," provides information to prepare for the installation of J2EE agents in the Policy Agent 2.2 software set and, after installation, to locate J2EE agent files required to configure J2EE agents and to perform other tasks.

Chapter 3, "Installing Policy Agent 2.2 for Apache Tomcat 6.0," provides instructions for installing Policy Agent 2.2 for Apache Tomcat 6.0.

Chapter 4, "Post-Installation Tasks of Policy Agent 2.2 for Apache Tomcat 6.0," provides information about J2EE agent configuration that is required for Policy Agent 2.2 for agents to function as intended.

Chapter 5, "Managing Policy Agent 2.2 for Apache Tomcat 6.0," provides information about the methods available for managing Policy Agent 2.2 for Apache Tomcat 6.0, with most of the information being applicable to all J2EE agents in the Policy Agent 2.2 software set.

Chapter 6, "Uninstalling Policy Agent 2.2 for Apache Tomcat 6.0," provides instructions for uninstalling Policy Agent 2.2 for Apache Tomcat 6.0.

Appendix A, "Silent Installation and Uninstallation of a J2EE Agent in Policy Agent 2.2," provides instructions for creating and using a script for automatic installation or uninstallation of a J2EE agent in the Policy Agent 2.2 software set.

Appendix B, "J2EE Agent `AMAgent.properties` Configuration File in Policy Agent 2.2," provides a list of the properties in the J2EE agent `AMAgent.properties` configuration file for Policy Agent 2.2 for Apache Tomcat 6.0 with most properties being applicable to all the J2EE agents in the Policy Agent 2.2 software set.

Appendix C, "Troubleshooting a J2EE Agent Deployment in Policy Agent 2.2," provides troubleshooting information about Sun Java System Policy Agent 2.2 for Apache Tomcat 6.0 that includes potential symptoms, causes, and solutions.

Related Books

Sun Microsystems server documentation sets, some of which are mentioned in this preface, are available at `http://docs.sun.com`. These documentation sets provide information that can be helpful for a deployment that includes Policy Agent.

Access Manager Documentation Set

Policy Agent2.2 was first introduced with Access Manager 7, but now also supports Access Manager 7.1. The information in the table that follows specifies documents in the Access Manager 7 documentation set, which is available at the following location:

`http://docs.sun.com/app/docs/coll/1292.1`

The Access Manager 7.1 documentation set is available at this location:

`http://docs.sun.com/app/docs/coll/1292.2`

TABLE P–1 Access Manager 7 2005Q4 Documentation Set

Title	Description
Sun Java System Access Manager 7 2005Q4 Release Notes	Available after the product is released. Contains last-minute information, including a description of what is new in this current release, known problems and limitations, installation notes, and how to report issues with the software or the documentation.
Sun Java System Access Manager 7 2005Q4 Technical Overview	Provides an overview of how Access Manager components work together to consolidate identity management and to protect enterprise assets and web-based applications. Explains basic Access Manager concepts and terminology

TABLE P–1 Access Manager 7 2005Q4 Documentation Set *(Continued)*

Title	Description
Sun Java System Access Manager 7 2005Q4 Deployment Planning Guide	Provides information about planning a deployment within an existing information technology infrastructure
Sun Java System Access Manager 7 2005Q4 Performance Tuning Guide	Describes how to tune Access Manager and its related components.
Sun Java System Access Manager 7 2005Q4 Administration Guide	Describes how to use Access Manager Console as well as how to manage user and service data via the command line.
Sun Java System Access Manager 7 2005Q4 Federation and SAML Administration Guide	Provides information about the features in Access Manager that are based on the Liberty Alliance Project and SAML specifications. It includes information on the integrated services based on these specifications, instructions for enabling a Liberty-based environment, and summaries of the application programming interface (API) for extending the framework.
Sun Java System Access Manager 7 2005Q4 Developer's Guide	Offers information on how to customize Access Manager and integrate its functionality into an organization's current technical infrastructure. Contains details about the programmatic aspects of the product and its API.
Sun Java System Access Manager 7 2005Q4 C API Reference	Provides summaries of data types, structures, and functions that make up the Access Manager public C APIs.
Sun Java System Access Manager 7 2005Q4 Java API Reference	Are generated from Java code using the JavaDoc tool. The pages provide information on the implementation of the Java packages in Access Manager.
Sun Java System Access Manager Policy Agent 2.2 User's Guide	Provides an overview of Policy Agent software, introducing web and J2EE agents. Also provides a list of web and J2EE agents currently available.

Updates to the *Release Notes* and links to modifications of the core documentation can be found on the Access Manager page at the Sun Java System 2005Q4 documentation web site. Updated documents are marked with a revision date.

Policy Agent 2.2 Documentation Set

Other Policy Agent guides, besides this guide, are available as described in the following sections:

- "*Sun Java System Access Manager Policy Agent 2.2 User's Guide*" on page 13
- "Other Individual Agent Guides" on page 13
- "Release Notes" on page 14

Sun Java System Access Manager Policy Agent 2.2 User's Guide

The *Sun Java System Access Manager Policy Agent 2.2 User's Guide* is available in two documentation sets: the Access Manager documentation set as described in Table P–1 and in the Policy Agent 2.2 documentation set as described in this section.

Other Individual Agent Guides

The individual agents in the Policy Agent 2.2 software set, of which this book is an example, are available on a different schedule than Access Manager itself. Therefore, documentation for Access Manager and Policy Agent are available in separate sets, except for the *Sun Java System Access Manager Policy Agent 2.2 User's Guide*, which is 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 in the Policy Agent 2.2 software set provides general information about web agents and installation, configuration, and uninstallation information for a specific web agent.

Each J2EE agent guide in the Policy Agent 2.2 software set provides general information about J2EE agents and installation, configuration, and uninstallation information for a specific J2EE agent.

The individual agent guides are listed along with supported server information in the following chapters of the *Sun Java System Access Manager Policy Agent 2.2 User's Guide*:

Web Agents	Chapter 2, "Access Manager Policy Agent 2.2 Web Agents: Compatibility, Supported Servers, and Documentation," in *Sun Java System Access Manager Policy Agent 2.2 User's Guide*
J2EE Agents	Chapter 3, "Access Manager Policy Agent 2.2 J2EE Agents: Compatibility, Supported Servers, and Documentation," in *Sun Java System Access Manager Policy Agent 2.2 User's Guide*

Release Notes

The *Sun Java System Access Manager Policy Agent 2.2 Release Notes* are available 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.

Sun Java Enterprise System Product Documentation

For useful information for related products, see the following documentation collections on the Sun Java Enterprise System documentation web site (`http://docs.sun.com/prod/entsys.05q4`)

- Sun Java System Directory Server:

  `http://docs.sun.com/coll/1316.1`
- Sun Java System Web Server:

  `http://docs.sun.com/coll/1308.1`
- Sun Java System Application Server:

  `http://docs.sun.com/coll/1310.1`
- Sun Java System Message Queue:

  `http://docs.sun.com/coll/1307.1`
- Sun Java System 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 Java System Services Suite
`http://www.sun.com/service/sunps/sunone/index.html`

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 Java System Access Manager Policy Agent 2.2 Guide for Apache Tomcat 6.0*, and the part number is 820-4802.

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 `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`

`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`**
		`Password:`

aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. Perform a *patch analysis*. Do *not* save the file. [Note that some emphasized items appear bold online.]

Shell Prompts in Command Examples

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

TABLE P–3 Shell Prompts

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

CHAPTER 1

Introduction to J2EE Agents for Policy Agent 2.2

The Sun Java™ System Access Manager Policy Agent 2.2 software set includes web agents and Java 2 Platform Enterprise Edition (J2EE) agents. This guide discusses J2EE agents, the functionality of which has increased for this release. This chapter provides a brief overview of J2EE agents in the 2.2 release as well as some concepts you need to understand before proceeding with a J2EE agent deployment. For a general introduction of agents, both J2EE agents and web agents, see *Sun Java System Access Manager Policy Agent 2.2 User's Guide*.

Deployment container:
: This term is used throughout this book to refer to a J2EE-compliant container that is either an application server or a portal server.

As was true in previous releases, J2EE agents enable deployment containers to enforce authentication and authorization using Sun Java System Access Manager 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 Access Manager)
- Single sign-on (SSO)

This chapter provides information about J2EE agents for the 2.2 release of Policy Agent as follows:

Uses of J2EE Agents

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

The way J2EE agents are used has not changed significantly in the 2.2 release. The agents perform more effectively and efficiently in this release, but the basic functions are the same.

J2EE agents help enable role-to-principal mapping for protected J2EE applications with Access Manager principals. Thus at runtime, when a J2EE policy is evaluated, it is done against the information available in Access Manager. 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 Access Manager based URL policies for enforcing access control over web resources hosted in the deployment container.

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

J2EE Agents and an Online Auction Application

Consider a web-based application that facilitates the auction of various kinds of merchandise between interested parties. A simple implementation for such an application will require the users to be in one of three abstract roles, namely Buyer, Seller, or Administrator. Buyers in this application will have access to web pages that display the listed auction items, whereas the Sellers may have access to web pages that allow them to list their merchandise for new auctions. The Administrators may have access to yet another set of web pages that allow them to finalize or cancel existing auctions in whatever state they may be in. Using the deployment descriptors, the application developer can express this intent by protecting such components using abstract security role names. These abstract role names in turn can be mapped to real principals in a J2EE agent. For example, the role Buyer may be mapped to an Access Manager role called Employee, the role Seller to an Access Manager role called Vendor, and the role Administrator to an Access Manager 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 Access Manager role called Vendor. Similarly, users Scott and Gina can place bids on this listed item only if they belong to the role called Buyer. 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 Access Manager. 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 Access Manager role called Staff, and the role Approver is mapped to an Access Manager 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 Access Manager 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 Access Manager 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 Access Manager. The deployment container then communicates with Access Manager 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 Access Manager to enforce J2EE security policies as well as Access Manager 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 Access Manager SDK for the deployment container. This SDK offers a rich set of APIs supported by Access Manager that can be used to create security-aware applications that are tailored to work in the security framework offered by Access Manager. For more information on how to use Access Manager SDK, see *Sun Java System Access Manager 7 2005Q4 Developer's Guide*.

What's New About J2EE Agents

J2EE agents offer greater functionality in the 2.2 release. Several important new features have been added as follows:

- "Removal of J2EE Agent Dependency on LDAP and on Administrative Accounts" on page 21
- "Enhanced J2EE Agent Installation Process" on page 22
- "J2EE Agent Coexistence With Access Manager" on page 23

- "J2EE Agent Support for Client Identification Based on Custom HTTP Headers" on page 24
- "J2EE Agent Specific Application for Housekeeping Tasks" on page 24
- "J2EE Agent URL Policy Enhancements" on page 25
- "J2EE Agent Support for Flexible User Mapping Mechanisms" on page 26
- "J2EE Agent Support for Fetching User Session Attributes" on page 27
- "J2EE Agent Support for Version Checking" on page 27
- "J2EE Agent Support for Not-Enforced IP List" on page 27
- "J2EE Agent Support for Custom Response Headers" on page 28
- "J2EE Agent Support for Application Logout Integration" on page 28
- "J2EE Agent Support for Application Specific Agent Filter Operation Modes" on page 29
- "J2EE Agent Support for Affinity-Based Login URL Selection" on page 29
- "J2EE Agent Support for a Sample Application" on page 29
- "J2EE Agent Backward Compatibility With Access Manager 6.3" on page 30

Removal of J2EE Agent Dependency on LDAP and on Administrative Accounts

In the 2.2 release, certain restrictions have been removed as follows:

- "Removal of J2EE Agent Dependency on LDAP" on page 21
- "Removal of J2EE Agent Dependency on Administrative Accounts" on page 22

Removal of J2EE Agent Dependency on LDAP

Unlike previous releases, J2EE agents in the Policy Agent 2.2 release do not use a direct LDAP connection. Instead, J2EE agents obtain support for their entire functionality by communicating with Access Manager solely with XML over HTTP.

Benefit - Removal of Dependency on LDAP: The benefit of not having an LDAP dependency includes greater flexibility and scalability of deployments. Since J2EE agents no longer depend on LDAP connections, they do not require the opening of LDAP communication ports in firewalls, which was a requirement with certain deployment scenarios in prior J2EE agent releases. With the LDAP dependency removed, the 2.2 release of J2EE agents requires fewer configuration changes during installation in protected regions, such as in a demilitarized zone (DMZ), giving more deployment flexibility and easing administrative overhead. Removal of the LDAP dependency also ensures that LDAP server resources are focused to support Access Manager instances. A focus on Access Manager instances facilitates the sizing process by eliminating considerations about the load that an agent would require. This makes the deployment easily scalable and more flexible and provides the optimal utilization of deployed resources.

Removal of J2EE Agent Dependency on Administrative Accounts

With the authorization of administrators now being handled by an *agent profile* account, the dependence on two administrative accounts, the `amAdmin` account and the `amldapuser` account, has been removed. Now, during installation, the agent installer prompts you for the agent profile account.

Benefit - Removal of Dependency on Administrative Accounts: The benefit of not using the `amAdmin` or `amldapuser` administrative accounts is greater security. The 2.2 release of J2EE agents depends solely on a limited agent profile. This dependence does not rely on the existence of sensitive account information in the agent deployment configuration.

Enhanced J2EE Agent Installation Process

Starting with this release of J2EE agents, the installation process includes the following features that allow for a smoother, less restrictive, more secure installation process and deployment:

- "J2EE Agent Support for Installation Using Non-Administrative User Accounts" on page 22
- "Secure Handling of Sensitive Information by J2EE Agents" on page 22
- "Self-Contained Installation of J2EE Agents" on page 23
- "J2EE Agent Support for Multiple Physical Installations" on page 23

J2EE Agent Support for Installation Using Non-Administrative User Accounts

The requirement in prior agent releases that the installation user have root (or Administrator) privileges has been removed. The agent can now be installed by any user regardless of access privileges.

Benefit - Support for Installation Using Non-Administrative User Accounts: The benefit of this feature is that it contributes to a more flexible installation process. Because privileged user accounts are not required, you can install agents in any directory location based on user preferences.

Secure Handling of Sensitive Information by J2EE Agents

All sensitive information, such as passwords, is now read from files. This information is not typed in clear text during an interactive session. Therefore, it is never displayed on the command line or in any logs. See "Using the Installation Program of Agent for Apache Tomcat 6.0" on page 61 for information about creating and using a password file during the agent installation process.

Benefit - Secure Handling of Sensitive Information: The benefit of this feature is increased security. With the use of this feature in the new installation process, the need for typing passwords in clear text on the console has been eliminated, thereby making the installation process less vulnerable to password theft.

Self-Contained Installation of J2EE Agents

All J2EE agent configuration and log files are generated and maintained within an agent's installation directory. This installation directory is referred to as the Policy Agent base directory. In code examples this directory is listed as such, *PolicyAgent-base*. For more information about the Policy Agent base directory, see "J2EE Agent Directory Structure in Policy Agent 2.2" on page 46. The distribution files for the 2.2 release of J2EE agents are provided to you in three formats. You can choose the format that best fits your needs: zip format, tar format, or a package format. The files are small in size since the installer for this release uses a simple configuration mechanism. In summary, when you unpack the binaries, the configuration and log files remain within the installation directory.

Benefit - Self-Contained Installation: This feature contributes to a more flexible installation process. The fact that an installation is self-contained facilitates the installation process and the subsequent administration process.

J2EE Agent Support for Multiple Physical Installations

There is no longer any restriction on using multiple different binaries of the same agent on the same machine.

Benefit - Support for Multiple Physical Installations: This feature contributes to a more flexible installation process. The fact that multiple binaries of the same agent can co-exist on the same machine allows for flexibility in securing complex environments where more than one server is installed on the same machine. Typically such environments are used for development purposes or for production using high capacity hardware systems.

J2EE Agent Coexistence With Access Manager

Starting with this release, you can deploy a J2EE agent on an instance of a deployment container where Access Manager has already been installed. This situation only applies when the J2EE agent and Access Manager both support the deployment container.

Note that Access Manager should be installed prior to the agent being installed.

Benefit - Coexistence With Access Manager: Certain situations benefit from having a J2EE agent and Access Manager on one machine, which is commonly desired for development environments and for environments where the deployment container is installed on a high capacity system which can support customer applications along with the Access Manager deployment.

J2EE Agent Support for Client Identification Based on Custom HTTP Headers

Starting with this release, J2EE agents can be configured to use custom HTTP headers to identify the remote client IP address and host name. This client IP address is used to validate an Access Manager session or to evaluate applicable policies.

Benefit - Support for Client Identification Based on Custom HTTP Headers: This feature is specially useful in situations where a proxy server exists between the remote client and the agent-protected server. In such situations a problem occurs in that the client address information carried within the request is replaced by the address information of the proxy server. This address replacement adversely affects session validations and policy evaluations, which depend upon the correct address information. However, when proxy servers can be configured to send the actual client address information in separate headers, then J2EE agents in the 2.2 release can use that information. In summary, this feature allows agents to use the actual client address information in this type of deployment as if the request were never intercepted by an intermediate proxy server.

J2EE Agent Specific Application for Housekeeping Tasks

Starting with this release of J2EE agents, a bundled application is available to perform housekeeping tasks on the deployment container.

This bundled application, when deployed on an agent-protected deployment container instance, expands the agent's functionality. For example, this bundled application allows the agent to receive notifications and to support cross-domain single sign-on. In previous releases, this functionality was tied to an application referred to as the *primary application*, which was secured by the agent.

Benefit - Agent Specific Application for Housekeeping Tasks: The benefit of including this application with J2EE agents is that previously imposed restrictions have been removed from the primary application. In prior releases of J2EE agents, the primary application had to support such housekeeping tasks, often requiring additional configuration. For instance, in prior releases, deploying the primary application with blind web-tier declarative security required changes to the agent configuration to ensure that the agent would function properly. This additional configuration is not necessary with the current release since the agent-specific application for housekeeping tasks takes care of all such functionality.

J2EE Agent URL Policy Enhancements

Starting with this release of J2EE agents, the following features are available that enhance Uniform Resource Locator (URL) policy:

- "Remote Policy Evaluation Failover in J2EE Agents" on page 25
- "Configurable Policy Evaluation Mechanism in J2EE Agents" on page 25
- "Composite Advice in J2EE Agents" on page 26
- "Policy Based Response Attributes in J2EE Agents" on page 26

The aforementioned features affect how agents enforce policy decisions. These features are described in the following paragraphs.

Remote Policy Evaluation Failover in J2EE Agents

J2EE agents can now leverage session failover functionality to ensure that remote policy evaluation failover can occur if an Access Manager instance becomes unavailable.

Benefit - Remote Policy Evaluation Failover: The benefit of this feature is the potential for a seamless user experience in the event of a failure or maintenance related outage of various servers within the Access Manager deployment.

Configurable Policy Evaluation Mechanism in J2EE Agents

Starting with this release, J2EE agents can be configured to use two different mechanisms to remotely evaluate policies as follows:

- The agent remotely requests policy evaluation for all resources applicable to a user within the agent's scope of protection.
- The agent remotely requests policy evaluation for the resource accessed by the user and not any other resources that the user might access later.

Benefit - Configurable Policy Evaluation Mechanism: The configurable policy evaluation mechanism has a variety of benefits depending on the situation. The very fact that you can configure the policy evaluation mechanism is beneficial in that it enables you to choose the mechanism that best fits your needs. Moreover, each mechanism has its advantages and disadvantages.

When configured to perform policy evaluation for all resources within an agent's scope of protection the first request is, by design, time consuming while subsequent requests are faster since the results get cached locally by the agent.

For the second mechanism, the first request is no slower or faster than subsequent requests. All requests are processed relatively quickly. However, this mechanism increases network communication between the agent and Access Manager. Furthermore, the two mechanism can produce different types of cache growth in agent memory and thus need to be evaluated closely to select the best option for your deployment. Key factors that could affect such a decision include the number of possible distinct resources and the number of policies applicable to an average user.

Composite Advice in J2EE Agents

In the 2.2 release, J2EE agents provide a composite advice feature. This feature allows the policy and authentication services of Access Manager to decouple the advice handling mechanism of the agents. This allows you to introduce and manage custom advices by solely writing Access Manager side plug-ins. Starting with this release, you are not required to make changes on the agent side. Such advices are honored automatically by the composite advice handling mechanism.

Benefit - Composite Advice: A benefit of composite advice is that you can incorporate a custom advice type without having to make changes to an agent deployment.

Policy Based Response Attributes in J2EE Agents

The policy-based response attribute feature allows the policy service to provide static and dynamic attributes based on the resource accessed by the user. These attributes can be made available to the agent protected application as HTTP headers, request attributes, or cookies.

Benefit - Policy Based Response Attributes: A benefit of the policy-based response attribute feature is that it provides more flexible support for application-specific agent customizations compared to prior Policy Agent releases, which only allowed for static profile attributes to be fetched.

J2EE Agent Support for Flexible User Mapping Mechanisms

Starting with this release, J2EE agents provide support for user mapping modes that have flexibility in the user names they choose. In prior releases, a user name had to be an Access Manager user ID. Now, user names can be chosen from a few different sources as long as the names are for authenticated users who have trusted identities. A trusted identity can be established on the agent-protected server for a security principal (or for an equivalent trusted identity of the user). This mechanism allows the agent to choose a user ID for the authenticated user from the user's profile attributes, the user's session properties, or an HTTP header accompanying the user request.

Benefit - Support for Flexible User Mapping Mechanisms: The main benefit of this feature is that it enables a J2EE agent to integrate with a greater number of applications. Some applications do not accept Access Manager user IDs as user names. J2EE agents can now integrate with those applications since Policy Agent 2.2 can be configured to provide different types of user names.

J2EE Agent Support for Fetching User Session Attributes

Before this release of J2EE agents, information for HTTP headers, request attributes, or cookies was retrieved, or *sourced*, solely from profile attributes. Now, this information can also be sourced from session properties.

Benefit - Support for Fetching User Session Attributes: The benefit of this feature is that session properties can be more effective for transferring information, especially dynamic information. Prior to this release, agents could only fetch users' profile attributes, which tend to be static attributes. However, session attributes allow applications to obtain dynamic user information when necessary.

J2EE Agent Support for Version Checking

Starting with this release of J2EE agents, you can easily check the exact version of the agent you are using, including build date, build number, and client SDK version. Prior to this release, administrators could not easily identify the build date of the agent they were using. Since code changes occur between build dates, identifying the exact build can be useful. For the details on how to check the version of an agent instance, see "`agentadmin --version`" on page 40.

Benefit - Support for Version Checking: The benefit of this feature is that it allows you to quickly check the version of the J2EE agent you are using, enabling you to determine which features and bug fixes are included.

J2EE Agent Support for Not-Enforced IP List

Starting with this release, J2EE agents support *not-enforced IP lists*. This new feature is similar to a pre-existing Policy Agent feature that also concerns not-enforced lists, specifically *not-enforced URI lists*.

The two features share similarities, but are really quite different. Again, the pre-existing feature supports not-enforced URI lists. With that feature, an agent always grants access to a URI that appears on a specified list in the J2EE agent `AMAgent.properties` configuration file. On the other hand, the new feature supports not-enforced IP lists. With this feature, an agent always grants access to resources when the request comes from a machine with an IP address that appears on a specified list in the J2EE agent `AMAgent.properties` configuration file.

With the new feature, when a request is made to access a resource, a J2EE agent determines the IP address of the machine where the request originated. The agent compares that IP address to all the addresses on the not-enforced IP list. If that address is on the list, then that request and all subsequent requests from that IP address are treated as if the resources requested are not enforced.

The not-enforced IP list can include exact IP addresses and IP addresses that use the asterisk, `*`, wildcard character to represent one or more characters.

Benefit - Support for Not-Enforced IP Lists: The benefit of this feature is that it allows clients on the not-enforced IP list to by-pass authentication and authorization requirements. This feature can be employed for administrative, troubleshooting, and testing purposes, too.

J2EE Agent Support for Custom Response Headers

Starting with this release, J2EE agents provide support for custom response headers. The agent can be configured so that custom response headers are set on every request. Such headers are defined statically in the J2EE agent `AMAgent.properties` configuration file and are honored on all enforced web resources as identified by the agent.

Benefit - Support for Custom Response Headers: The benefit of this feature is that it enables instructions, by way of custom response headers, to be sent to a client or any intermediate entity between the agent-protected server and the client browser. For example, a custom response header could be sent instructing an intermediate proxy server not to cache server responses to client requests.

J2EE Agent Support for Application Logout Integration

Starting with this release, J2EE agents can be configured to identify an application logout event and to synchronize the event with the Access Manager logout. The agent can identify the logout of an application based on preconfigured information sent with a request as follows:

- The request URI
- A query parameter sent with the request
- A request parameter sent within the request body

Benefit - Support for Application Logout Integration: The benefit of this logout integration feature is that end users can potentially perform a global log out by simply logging out of an application. Also, J2EE agents, starting with the 2.2 release, are very flexible in terms of the variety of ways they can identify the log out of an application. This flexibility facilitates the integration between J2EE agents and applications since applications will rarely need any modifications to enable them to match any requirements of the agent.

J2EE Agent Support for Application Specific Agent Filter Operation Modes

The application-specific filter operation mode mechanism allows different applications to use different levels of protection as necessary. Different filter operation modes provide different levels of functionality, thus enabling the selection of the best mode for each protected application.

Benefit - Support for Application Specific Agent Filter Operation Modes: This feature provides customized protection for every application. For example, if two applications are deployed where one uses J2EE policies and the other does not, the agent can be configured to use different filter modes for each application. The agent will provide support for J2EE policies, but only for the application that has such policies. The results is the optimal use of system resources since the agent does not enforce the evaluation of J2EE policies for the application that does not have any J2EE policies.

J2EE Agent Support for Affinity-Based Login URL Selection

Starting with this release, J2EE agents support a prioritized (or an *affinity-based*) selection of login URLs for authenticating users. End users are directed to the URL highest on the list if it is available. If not, the second URL on the list is targeted. If a URL higher on the list becomes available again, the agent switches to that URL.

You can disable this affinity-based selection process if desired, which would allow you to use a round-robin selection scheme.

Benefit - Support for Affinity-Based Login URL Selection: This feature is used when two or more Access Manager instances are deployed in geographically distant locations. To best manage the authentication process, you can give the highest priority to URLs of locally available Access Manager instances, resulting in faster response times.

J2EE Agent Support for a Sample Application

Starting with this release, the J2EE agents provide a bundled sample application to demonstrate the key features and functionality of the agent. Some of the features demonstrated are:

- J2EE declarative security
- J2EE programmatic security
- URL policy-based access control deployed on an agent-protected deployment container

For more information about the sample application, see "The Sample Application" on page 31 and see the list following Table 2–2 for information about locating the `sampleapp` directory within the J2EE agent base directory. The `sampleapp` directory includes instructions on how to use the sample application.

Benefit - Support for a Sample Application: The sample application illustrates how applications need to be configured to take advantage of the protection that agents provide. The sample application also demonstrates how key functionality, such as support for J2EE security and Access Manager based URL policies, can be used.

J2EE Agent Backward Compatibility With Access Manager 6.3

Policy Agent 2.2 is backward compatible with Access Manager 6.3 Patch 1 or greater.

Note – Policy Agent 2.2 is only compatible with Access Manager 6.3 when the Access Manager patch has been applied.

By default, J2EE agents in the Policy Agent 2.2 release work with Access Manager 7. While the 2.2 release of Policy Agent requires some configuration to work with Access Manager 6.3 Patch 1 or greater, the amount of configuration required is relatively limited. For more information, see "Installing and Configuring the Apache Tomcat 6.0 Agent With Access Manager 6.3" on page 51.

Be aware that Policy Agent 2.2 takes advantage of certain features that exist in Access Manager 7 that do not exist in Access Manager 6.3, such as "composite advices," "policy-based response attributes," and others.

Information About Using J2EE Agents in Policy Agent 2.2

This section provides information about J2EE agents in Policy Agent 2.2 that will help you install and get accustomed to the product. J2EE agents have undergone some major changes for the 2.2 release that affect how you interact with them. Read the following subsections to understand the more significant ways that changes made in the 2.2 release affect the manner in which you use a J2EE agent.

Enhanced Installation Process for J2EE Agents in Policy Agent 2.2

The installation process for J2EE agents is quite different for the 2.2 release. This guide provides you with a chapter that explains all the details necessary for understanding this process, from unpacking the J2EE agent binaries to installation related commands, to the directory structure of J2EE agents once the binaries are unpacked. See Chapter 2, "Vital Installation Information for a J2EE Agent in Policy Agent 2.2."

Increased Functionality of the `agentadmin` Program for J2EE Agents in Policy Agent 2.2

The `agentadmin` program is a required tool for the 2.2 release of J2EE agents. The functionality has increased significantly. The most basic of tasks, such as installation and uninstallation can only be performed with this tool.

For detailed information on installation related tasks performed with this program, see "Role of the `agentadmin` Program in a J2EE Agent for Policy Agent 2.2" on page 34.

For information on all the tasks performed with this program, see "Key Features and Tasks Performed With the J2EE `agentadmin` Program" on page 105.

The Sample Application

Deploy, and interact with the sample application included with Policy Agent 2.2. 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 2–2. 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`

CHAPTER 2

Vital Installation Information for a J2EE Agent in Policy Agent 2.2

To make the installation process of a J2EE agent in Policy Agent 2.2 simple, essential information needed for the installation is provided in this chapter.

This chapter applies to all the J2EE agents in the Policy Agent 2.2 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 2.2 for Sun Java System Application Server 8.1. These examples are provided to illustrate general format. Replace J2EE agent specific information where necessary.

When you are comfortable with the information presented in this chapter, move on to the installation as described in Chapter 3, "Installing Policy Agent 2.2 for Apache Tomcat 6.0."

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 Access Manager 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 2.2" on page 34
- "Role of the `agentadmin` Program in a J2EE Agent for Policy Agent 2.2" on page 34
- "J2EE Agent Directory Structure in Policy Agent 2.2" on page 46

- "Installing and Configuring the Apache Tomcat 6.0 Agent With Access Manager 6.3" on page 51
- "Creating a J2EE Agent Profile" on page 52

Format of the Distribution Files for a J2EE Agent Installation in Policy Agent 2.2

The distribution files for a J2EE agent in Policy Agent 2.2 are provided to you in `.zip` archive. For example

appserver_version`_agent.zip`

where *appserver_version* is a place holder that represents the specific deployment container and deployment container version you are unpacking.

▼ To Unpack a `.zip` Compressed file of a J2EE Agent in Policy Agent 2.2

The instructions in this task are not specific to the J2EE agent described in this guide. For the specific instructions, see "To Prepare to Install Agent for Apache Tomcat 6.0" on page 57.

- **Unzip the `.zip` file using the appropriate utility or command for your platform. For example, on Solaris systems, issue the following command:**

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

Role of the `agentadmin` Program in a J2EE Agent for Policy Agent 2.2

The `agentadmin` program is a required install and configuration tool for the 2.2 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.

TABLE 2–1 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 2–1 Command Format: `agentadmin --install`

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

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

EXAMPLE 2–1 Command Format: `agentadmin --install` *(Continued)*

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 2–2 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 2–3 Command Format: `agentadmin --uninstall`

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

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

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

`--saveResponse`	Use this argument to save all supplied responses to a state file, or response file, represented as *filename* in command examples. The response file, or state file, can then be used for silent uninstallations.
`--useResponse`	Use this argument to uninstall a J2EE agent in silent mode as all uninstaller prompts are answered with responses previously saved to a response file, represented as *filename* in command examples. When this argument is used, the uninstaller runs in non-interactive mode. At which time, user interaction is not required.
filename	Use this argument to specify the name of a file that will be created as part of the processing of this command. This file stores your responses when this argument is used in conjunction with the `--saveResponse` argument and provides your responses when this argument is used in conjunction with the `--useResponse` argument.

EXAMPLE 2–4 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.

EXAMPLE 2–4 Command Usage: `agentadmin --uninstall` *(Continued)*

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

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

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

`agentadmin --listAgents`

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

EXAMPLE 2–5 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 2–6 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 8.1, 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:
```

EXAMPLE 2–6 Command Usage: `agentadmin --listAgents` *(Continued)*

```
/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 2–7 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 2–8 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 8.1, 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
```

EXAMPLE 2–8 Command Usage: `agentadmin --agentInfo` *(Continued)*

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 2–9 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 2–10 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 8.1, 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 Access Manager 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 2–11 Command Format: `agentadmin --encrypt`

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

```
./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 for a J2EE agent instance be present in the `AMAgent.properties` configuration file of that specific J2EE agent instance.
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. For more information, see "Preparing to Install Agent for Apache Tomcat 6.0" on page 57

EXAMPLE 2–12 Command Usage: `agentadmin --encrypt`

Issuing the `agentadmin` command with the `--encrypt` option enables you to change the password for an existing agent profile in Access Manager 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 Access Manager. Once the agent profile for a specific agent has been created in Access Manager, the installer enters the Policy Agent profile name and encrypted password in the respective J2EE agent

EXAMPLE 2–12 Command Usage: `agentadmin --encrypt` *(Continued)*

`AMAgent.properties` configuration file for the agent instance. If you choose a new password for the Policy Agent profile, encrypt it and enter that encrypted password in the J2EE agent `AMAgent.properties` configuration file with the following property:

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

agentadmin --getEncryptKey

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

EXAMPLE 2–13 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 2–14 Command Usage: `agentadmin --getEncryptKey`

This option may be used in conjunction with the `--encrypt` option to encrypt and decrypt sensitive information in the J2EE agent `AMAgent.properties` configuration file. Issuing the `agentadmin` command with the `--getEncryptKey` option generates a new encryption key for the J2EE agent.

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

```
./agentadmin -getEncryptKey


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

The encryption key is stored in the J2EE agent `AMAgent.properties` configuration 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 J2EE agent `AMAgent.properties` configuration file stores the encryption key:

```
com.sun.identity.client.encryptionKey
```

EXAMPLE 2–14 Command Usage: `agentadmin --getEncryptKey` *(Continued)*

For example, using the encryption key example provided previously, updating the encryption key value in the J2EE agent `AMAgent.properties` configuration file could appear as follows:

```
com.sun.identity.client.encryptionKey = k1441g4EejuOgsPlFOSg+m6P5x7/G9rb
```

Once you have updated the J2EE agent `AMAgent.properties` configuration 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 2–15 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 2–16 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 2–17 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:

userName	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 *IdType* parameter. Therefore, if the identity type is for a user, this *userName* parameter would be the name of that user.
IdType	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.
realmName	Use this third parameter to specify the name of the default organization of the Access Manager 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 Access Manager 7. Do not use this option with earlier versions of Access Manager, such as version 6.3. If the application is deployed with Access Manager 6.3 principals or roles, replace the role-to-principal mappings with the distinguished name (DN) of the user in Access Manager 6.3.

EXAMPLE 2–18 Command Usage: `agentadmin --getUuid`

In Access Manager 7, issuing the `agentadmin` command with the `--getUuid` option retrieves the universal ID of any identity type in Access Manager 7.

If you run the agent in J2EE_POLICY mode, you must repackage the web applications with Access Manager 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 2–19 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 2–20 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.
--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 2–21 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 2–22 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.

A 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 2.2

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 2.2

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 2.2 for Sun Java System Application Server 8.1, 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. To see the preceding directory name specific to your J2EE agent, see Example 3–1.

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 2–23. For information about choosing a directory in which to unpack the J2EE agent binaries, see "Format of the Distribution Files for a J2EE Agent Installation in Policy Agent 2.2" on page 34.

EXAMPLE 2–23 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 2.2 for Sun Java System Application Server 8.1:

Agent-HomeDirectory`/j2ee_agents/am_as81_agent`

For other J2EE agents, the directory names are different, but the naming format is the same. To see the preceding path name specific to your J2EE agent, see Example 3–1. References in this book to the *PolicyAgent-base* directory are references to the preceding path.

Inside the J2EE Agent Base Directory in Policy Agent 2.2

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 2–2.

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 2–2 Example of Policy Agent Base Directory for a J2EE Agent

Directory Contents: Files and Subdirectories	
`LICENSE.TXT`	`jce`

TABLE 2–2 Example of Policy Agent Base Directory for a J2EE Agent *(Continued)*

Directory Contents: Files and Subdirectories	
`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 2.2. 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 31.

`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 2.2" on page 34.

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

This directory also contains the installation log file. For the 2.2 release of Policy Agent, log information is stored in the installation log file after you install a J2EE agent instance. The following is the location of this log file:

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

`lib`	The `lib` directory has a list of all the agent libraries that are used by the installer as well as the agent run time.
`locale`	This directory has all the agent installer information as well as agent run time specific locale information pertaining to the specific agent.
`data`	This directory has all the installer specific data.

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

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

PolicyAgent-base/*AgentInstance-Dir*

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

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

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

  Note – Agent-specific debug information is not stored in this directory when the J2EE agent and Access Manager are installed on the same deployment container. However, the J2EE agent and Access Manager 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 Access Manager directory:

  `/var/opt/SUNWam/debug`

`config` This directory contains the J2EE agent `AMAgent.properties` configuration file that is specific to the agent instance. Each J2EE agent can be configured by a unique instance of the J2EE agent `AMAgent.properties` configuration file. This file holds the key to the agent behavior at runtime.

Installing and Configuring the Apache Tomcat 6.0 Agent With Access Manager 6.3

Although the Tomcat 6.0 agent is intended to be used with Access Manager 7.1, you can configure the agent to function with Access Manager 6 2005Q1 (6.3) patch 1 or later. However, some of the Access Manager 7.1 features, such as composite advices and policy-based response attributes, are not available in Access Manager 6.3.

Caution – For the Tomcat 6.0 agent to function properly with Access Manager 6.3, patch 1 or greater must be applied to the Access Manager 6.3 instance.

▼ To Install and Configure the Tomcat 6.0 Agent With Access Manager 6.3

1. **Ensure that the Access Manager 6.3 instance has been updated with patch 1 or later.**

2. **Create an agent profile in the Access Manager 6.3 Console for the Tomcat 6.0 agent.**

   Save the agent profile information to use during agent installation in the next step. For information about creating the agent profile in Access Manager 6.3, see Chapter 4, Identity Management, in the *Sun Java System Access Manager 6 2005Q1 Administration Guide*.

3. **Install the Tomcat 6.0 agent, providing details for the Access Manager 6.3 instance.**

   For more information, see Chapter 3, "Installing Policy Agent 2.2 for Apache Tomcat 6.0."

4. **Change to the *PolicyAgent-base*/`lib` directory.**

5. **Download the `amclientsdk63.jar` and `fmclientsdk.jar` files to the *PolicyAgent-base*/`lib` directory from the OpenSSO Project site:**

   `https://opensso.dev.java.net/public/use/stablebuilds.html`

6. **Edit the `classpath` in the `setAgentEnv_`*server-instance*`.sh` UNIX script or `setAgentEnv_`*server-instance*`.cmd` Windows script to specify the files you downloaded in the previous step.**

   Important: You must remove *PolicyAgent-base*`/lib/openssoclientsdk.jar;` from the `classpath`.

7 **In the `web.xml` file of a application that needs to be protected by the agent, the roles should be defined as `"cn=role_name,dc=domain"` instead of `"id=role_name,ou=role,dc=domain"`.**

For example: `"cn=manager,dc=example,dc=com"` instead of `"id=manager,ou=role,dc=example,dc=com"`.

Creating a J2EE Agent Profile

Caution – Creating a J2EE agent profile in Access Manager 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 Access Manager prevents the J2EE agent from authenticating or having any further communication with Access Manager.

J2EE agents work with Access Manager 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 Access Manager 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 Access Manager 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 Access Manager Console. Therefore, tasks related to the agent profile are discussed in Access Manager 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 Access Manager Console, see the following section of the respective document: "Agents" in *Sun Java System Access Manager 7 2005Q4 Administration Guide*.

▼ To Create an Agent Profile

Perform the following tasks in Access Manager 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.**

CHAPTER 3

Installing Policy Agent 2.2 for Apache Tomcat 6.0

Sun Java™ System Access Manager Policy Agent 2.2 for Apache Tomcat 6.0, as with all J2EE agents in the 2.2 release of Policy Agent, is installed from the command line using the `agentadmin` program. For more information about the tasks you can perform with the `agentadmin` program, see "Role of the `agentadmin` Program in a J2EE Agent for Policy Agent 2.2" on page 34.

Before reading this chapter or performing any of the tasks described within, thoroughly review Chapter 2, "Vital Installation Information for a J2EE Agent in Policy Agent 2.2," since various key concepts are introduced in that chapter.

This chapter is organized into the following sections:

- "Installation Related Information About Agent for Apache Tomcat 6.0" on page 56
- "Preparing to Install Agent for Apache Tomcat 6.0" on page 57
- "Launching the Installation Program of Agent for Apache Tomcat 6.0" on page 60
- "Using the Installation Program of Agent for Apache Tomcat 6.0" on page 61

Before describing any task, this chapter provides you with installation-related information specific to Apache Tomcat 6.0. The subsequent sections lead you through the pre—installation and installation steps and describe how to view the installation log files. First, perform the pre-installation (preparation) steps. Then, perform the installation, itself. The installation process has two phases. The first phase of the installation includes launching the installation program, which requires a directory to already have been selected for the agent files. The second phase of the installation involves interacting with the installation program. During this phase, the program prompts you step by step to enter information. Accompanying the prompts, are explanations of the type of information you need to enter. After you complete the installation, you can look at the installation log files.

Once you have completed the steps described in this chapter, complete the applicable post-installation tasks described in Chapter 4, "Post-Installation Tasks of Policy Agent 2.2 for Apache Tomcat 6.0."

Installation Related Information About Agent for Apache Tomcat 6.0

The following sections provide important information about Policy Agent 2.2 for Apache Tomcat 6.0 needed before you install the agent.

Supported Platforms and Compatibility of Agent for Apache Tomcat 6.0

The following sections provide information about the supported platforms of Policy Agent 2.2 for Apache Tomcat 6.0 as well as the compatibility of this agent with Access Manager.

Platform and Version Support of Agent for Apache Tomcat 6.0

The following table presents the platforms supported by Policy Agent 2.2 for Apache Tomcat 6.0.

TABLE 3–1 Platform and Version Support of Agent for Apache Tomcat 6.0

Agent for	Supported Policy Agent Version	Supported Access Manager Versions	Supported Platforms
Apache Tomcat 6.0	Version 2.2	Version 6.3 Patch 1 or greater Version 7 Version 7.1	Solaris™ Operating System (OS) for the SPARC® platform, versions 9 and 10 Solaris (OS) for x86 platforms, versions 9, and 10 Red Hat Enterprise Linux Advanced Server 4.0 and 5.0 Windows 2003, Enterprise Edition Windows 2003, Standard Edition

Compatibility of Agent for Apache Tomcat 6.0 With Access Manager

Compatibility of Policy Agent 2.2 With Access Manager 7 and Access Manager 7.1.

All agents in the Policy Agent 2.2 release are compatible with Access Manager 7 and Access Manager 7.1. Compatibility applies to both of the available modes of Access Manager: Realm Mode and Legacy Mode.

Install the latest Access Manager patches to ensure that all enhancements and fixes are applied. For an example of Access Manager patches that can be installed, see the compatibility information discussed in *Sun Java System Access Manager Policy Agent 2.2 Release Notes*.

Compatibility of Policy Agent 2.2 With Access Manager 6.3

Most agents in Policy Agent 2.2 are also compatible with Access Manager 6.3 Patch 1 or greater. However, certain limitations apply. For more information, see "J2EE Agent Backward Compatibility With Access Manager 6.3" on page 30.

Preparing to Install Agent for Apache Tomcat 6.0

Follow the specific steps outlined in the following section before you install the agent to reduce the chance of complications occurring during and after the installation.

▼ To Prepare to Install Agent for Apache Tomcat 6.0

Perform the following pre-installation steps:

1. **Ensure that the agent distribution files are properly unzipped on the Apache Tomcat 6.0 instance as described in the substeps that follow:**

   a. **Create a directory in which to download the** `tomcat_v6_agent.zip` **file.**

      For example:

      ```
      Agent_Home
      ```

   b. **Download the** `tomcat_v6_agent.zip` **file to the newly created directory.**

   c. **From the newly created directory, unzip the** `tomcat_v6_agent.zip` **file using the appropriate utility or command for your platform.**

      For example, on Solaris systems, issue the following:

      ```
      unzip tomcat_v6_agent.zip
      ```

2. **Ensure that Policy Agent 2.2 for Apache Tomcat 6.0 is supported on the desired platform as listed in "Supported Platforms and Compatibility of Agent for Apache Tomcat 6.0" on page 56.**

3. **Install Apache Tomcat 6.0 if not already installed.**

Caution – While the present release of Agent for Apache Tomcat 6.0 supports the `.exe` extension of the Apache Tomcat 6.0 bits, an extra task is required to make the installation work. For those instructions, see "(Conditional) To Use the `.exe` Version of Apache Tomcat 6.0 Server" on page 59.

The extra task is unnecessary if you use the `.zip` files or `.gz` files. For example, for version 6.0.14, you could download either of the following compressed files and perform the installation without having to implement the extra task:

```
apache-tomcat-6.0.14.zip
```

```
apache-tomcat-6.0.14.tar.gz
```

Compressed files for Apache Tomcat 6.0 as well as installation-related documentation are available at the Apache web site at `http://apache.org/`

4 **(Conditional) If the Apache Tomcat 6.0 instance on which you are about to install the agent is running, shut it down.**

5 **Create a valid agent profile in Access Manager Console if one has not already been created.**

For information on how to create an agent profile, see "Creating a J2EE Agent Profile" on page 52.

To avoid a misconfiguration of the agent, ensure that you know the exact ID and password used to create the agent profile. You must enter the agent profile password correctly in the next step and you must enter the agent profile ID correctly when installing the agent.

6 **Create a text file and add the agent profile password to that file.**

Ensure that this file is located in a secure directory of your choice. You will refer to this file during the agent installation process.

With the agent profile password in this file, stored in a secure location, you do not need to enter sensitive information in the console. A valid password file can have only one line that contains the agent profile password.

7 **(On Red Hat Enterprise Linux Advanced Server only) Ensure that the Sun Microsystems `java` file is configured to be retrieved instead of the Red Hat `java` file.**

You can achieve this using various methods. Pick the method of choice. The following is an example of how you can accomplish this:

Set the `JAVA_HOME` environment variable to point to the Sun Microsystems `java` file. The location varies, but the following example illustrates a feasible location for this file:

```
/share/builds/components/jdk/1.5.0_06/Linux
```

▼ (Conditional) To Use the `.exe` Version of Apache Tomcat 6.0 Server

As described in "To Prepare to Install Agent for Apache Tomcat 6.0" on page 57, you can install Agent for Apache Tomcat 6.0 on a `.exe` version of Apache Tomcat 6.0. However, you must implement the steps in this task prior to installing the agent for the deployment to work.

The Apache Tomcat 6.0 bits with the `.exe` extension do not contain certain files (scripts, JAR files, etc.) required by the agent. Therefore, you must copy the required files from the `.zip` bundle to the Apache Tomcat 6.0 installation base.

1 **Download the `.exe` and the `.zip` versions of Apache Tomcat 6.0.**

For example, you could download the two following Apache Tomcat 6.0 versions:

- `apache-tomcat-6.0.14.exe`
- `apache-tomcat-6.0.14.zip`

2 **Install Apache Tomcat 6.0 using the `.exe` version.**

If the version of Apache Tomcat 6.0 is indeed `6.0.14`, the following would be a feasible location for `CATALINA_HOME`:

```
/opt/apache-tomcat-6.0.14
```

3 **In a directory separate from the `.exe` distribution, unzip the `apache-tomcat-6.0.14.zip` file.**

4 **Copy the following scripts from the unzipped bundle to the bin directory of the installation (`${CATALINA_HOME}/bin`):**

```
catalina-tasks.xml
catalina.bat
catalina.sh
commons-daemon.jar
cpappend.bat
digest.bat
digest.sh
jsvc.tar.gz
service.bat
setclasspath.bat
setclasspath.sh
shutdown.bat
shutdown.sh
startup.bat
startup.sh
tomcat-native.tar.gz
tool-wrapper.bat
tool-wrapper.sh
```

```
version.bat
version.sh
```

Next Steps

Note – After you have implemented this task, the best practice is to start and stop the Apache Tomcat 6.0 instance using the command line since problems have occurred in this scenario when attempting to stop and start the server via services.

At this point, you can install Agent for Apache Tomcat 6.0.

Launching the Installation Program of Agent for Apache Tomcat 6.0

Once you have performed all the pre-installation steps, you can launch the installation program as described in the following subsection.

▼ To Launch the Installation Program of Agent for Apache Tomcat 6.0

To launch the installation program, perform the following steps:

1. **Change to the following directory:**

   PolicyAgent-base`/bin`

   This directory contains the `agentadmin` program, which is used for installing a J2EE agent and for performing other tasks. For more information on the `agentadmin` program, see "Role of the `agentadmin` Program in a J2EE Agent for Policy Agent 2.2" on page 34.

2. **Issue the following command:**

   ```
   ./agentadmin --install
   ```

3. **(Conditional) If you receive license agreement information, accept or reject the agreement prompts. If you reject any portion of the agreement, the program will end.**

   The license agreement is displayed only during the first run of the `agentadmin` program.

Using the Installation Program of Agent for Apache Tomcat 6.0

After you issue the `agentadmin` command and accept the license agreement (if necessary) the installation program appears, prompting you for information.

The steps in the installation program are displayed in this section in an example interaction. Your answers to prompts can differ slightly or greatly from this example depending upon your specific deployment. In the example, most of the defaults have been accepted. This example is provided for your reference and does not necessarily indicate the precise information you should enter.

The following bulleted list provides key points about the installation program.

- Each step in the installation program includes an explanation that is followed by a more succinct prompt.
- For most of the steps you can type any of the following characters to get the results described:

`?`	Type the question mark to display Help information for that specific step.
`<`	Type the left arrow symbol to go back to the previous interaction.
`!`	Type the exclamation point to exit the program.

- Most of the steps provide a default value that can be accepted or replaced. If a default value is correct for your site, accept it. If it is not correct, enter the correct value.

About Installation Prompts in Agent for Apache Tomcat 6.0

The following list provides information about specific prompts in the installation. Often the prompt is self explanatory. However, at other times you might find the extra information presented here to be very helpful. This extra information is often not obvious. Study this section carefully before issuing the `agentadmin --install command.`

The Deployment URI for the Agent Application
: The deployment URI for the agent application is required for the agent to perform necessary housekeeping tasks such as registering policy and session notifications, legacy browser support, and CDSSO support. Accept `/agentapp` as the default value for this interaction. Once the installation is completed, browse the directory *PolicyAgent-base*`/etc`. Use the `agentapp.war` file to deploy the agent application in the application container. Please note that the deployment URI for agent application during install time should match the deployment URI for the same application when deployed in the J2EE container.

The Encryption Key

This key is used to encrypt sensitive information such the passwords. The key should be at least 12 characters long. A key is generated randomly and provided as the default. You can accept the random key generated by the installer or create your own using the `.agentadmin --getEncryptKey` command.

For information about creating a new encryption key, see "`agentadmin --getEncryptKey`" on page 42.

The Agent Profile Name

An agent profile should have been created as a pre-installation step. The creation of the agent profile is mentioned in that section. For the pre-installation steps, see "Preparing to Install Agent for Apache Tomcat 6.0" on page 57. For the actual information on creating an agent profile, see "Creating a J2EE Agent Profile" on page 52.

In summary, the J2EE agent communicates with Access Manager with a specific ID and password created through an agent profile using Access Manager Console. For J2EE agents, the creation of an agent profile is mandatory. Access Manager uses the agent profile to authenticate an agent. This is part of the security infrastructure.

The J2EE Password File

The J2EE password file should have been created as a pre-installation step. For the pre-installation steps, see "Preparing to Install Agent for Apache Tomcat 6.0" on page 57.

When the installation program prompts you for the password for the agent, enter the fully qualified path to this password file.

After you have completed all the steps, a summary of your responses appears followed by options that allow you to navigate through those responses to accept or reject them.

When the summary appears, note the agent instance name, such as `agent-001`. You might be prompted for this name during the configuration process.

About the options, the default option is 1, Continue with Installation.

- If you are satisfied with the summary, choose 1 (the default).
- If you want to edit input from the last interaction, choose 2.
- If you want to edit input starting at the beginning of the installation program, choose 3.
- If you want to exit the installation program without installing, choose 4.

You can edit your responses as necessary, return to the options list, and choose option 1 to finally process your responses.

Example of Installation Program Interaction in Agent for Apache Tomcat 6.0

The following example is a sample installation snapshot of Policy Agent 2.2 for Apache Tomcat 6.0. By no means does this sample represent a real deployment scenario.

The section following this example, "Implications of Specific Deployment Scenarios in Agent for Apache Tomcat 6.0" on page 66, provides a short explanation about installing a J2EE agent on multiple Apache Tomcat 6.0 instances. If your deployment includes multiple instances of the deployment container, you might want to review that section before proceeding with the agent installation. See "Installing a J2EE Agent on Multiple Apache Tomcat 6.0 Instances" on page 66.

```
************************************************************************
Welcome to the Access Manager Policy Agent for Apache Tomcat 6.0 Servlet/JSP
Container

************************************************************************



Enter the complete path to the directory which is used by Tomcat Server to
store its configuration Files. This directory uniquely identifies the
Tomcat Server instance that is secured by this Agent.
[ ? : Help, ! : Exit ]
Enter the Tomcat Server Config Directory Path
[/opt/apache-tomcat-6.0.14/conf]:


Enter the fully qualified host name of the server where Access Manager
Services are installed.
[ ? : Help, < : Back, ! : Exit ]
Access Manager Services Host: amHost.example.com


Enter the port number of the Server that runs Access Manager Services.
[ ? : Help, < : Back, ! : Exit ]
Access Manager Services port [80]:


Enter http/https to specify the protocol used by the Server that runs Access
Manager services.
[ ? : Help, < : Back, ! : Exit ]
Access Manager Services Protocol [http]:


Enter the Deployment URI for Access Manager Services.
[ ? : Help, < : Back, ! : Exit ]
```

```
Access Manager Services Deployment URI [/amserver]:


Enter the fully qualified host name on which the Application Server
protected by the agent is installed.
[ ? : Help, < : Back, ! : Exit ]
Enter the Agent Host name: agentHost.example.com


$CATALINA_HOME environment variable is the root of the tomcat
installation.
[ ? : Help, < : Back, ! : Exit ]
Enter the $CATALINA_HOME environment variable: /opt/apache-tomcat-6.0.14/


Choose yes to deploy the policy agent in the global web.xml file.
[ ? : Help, < : Back, ! : Exit ]
Install agent filter in global web.xml ? [true]:


Enter the preferred port number on which the application server provides its
services.
[ ? : Help, < : Back, ! : Exit ]
Enter the port number for Application Server instance [80]:


Select http or https to specify the protocol used by the Application server
instance that will be protected by Access Manager Policy Agent.
[ ? : Help, < : Back, ! : Exit ]
Enter the Preferred Protocol for Application Server instance [http]:


Enter the deployment URI for the Agent Application. This Application is used
by the agent for internal housekeeping.
[ ? : Help, < : Back, ! : Exit ]
Enter the Deployment URI for the Agent Application [/agentapp]:


Enter a valid Encryption Key.
[ ? : Help, < : Back, ! : Exit ]
Enter the Encryption Key [kPd9i5QsulOzHqOT9q1vvzuHC2RNo9Sx]:


Enter a valid Agent profile name. Before proceeding with the agent
installation, please ensure that a valid Agent profile exists in Access
Manager.
[ ? : Help, < : Back, ! : Exit ]
Enter the Agent Profile name: MyAgent
Enter the path to a file that contains the password to be used for identifying
the Agent.
[ ? : Help, < : Back, ! : Exit ]
Enter the path to the password file: /tmp/agentpass.txt



-----------------------------------------------
SUMMARY OF YOUR RESPONSES
-----------------------------------------------
Tomcat Server Config Directory : /opt/apache-tomcat-6.0.14/conf
Access Manager Services Host : amHost.example.com
Access Manager Services Port : 80
Access Manager Services Protocol : http
Access Manager Services Deployment URI : /amserver
Agent Host name : nicp226.india.sun.com
$CATALINA_HOME environment variable : /opt/apache-tomcat-6.0.14/
Tomcat global web.xml filter install : true
Application Server Instance Port number : 80
Protocol for Application Server instance : http
Deployment URI for the Agent Application : /agentapp
Encryption Key : kPd9i5QsulOzHqOT9q1vvzuHC2RNo9Sx
Agent Profile name : MyAgent
Agent Profile Password file name : /tmp/agentpass.txt

Verify your settings above and decide from the choices below.
1. Continue with Installation
2. Back to the last interaction
3. Start Over
4. Exit
Please make your selection [1]: 1

Updating the /opt/apache-tomcat-6.0.14/bin/setclasspath.sh script with
the Agent classpath ...DONE.

Creating directory layout and configuring Agent file for Agent_001
instance ...DONE.

Reading data from file /tmp/agentpass.txt and encrypting it ...DONE.

Generating audit log file name ...DONE.

Creating tag swapped AMAgent.properties file for instance Agent_001 ...DONE.

Creating a backup for file /opt/apache-tomcat-6.0.14/conf/server.xml ...DONE.

Creating a backup for file /opt/apache-tomcat-6.0.14/conf/web.xml ...DONE.
```

```
Adding SJS Tomcat Agent Realm to Server XML file :
/opt/apache-tomcat-6.0.14/conf/server.xml ...DONE.

Adding filter to Global deployment descriptor file :
/opt/apache-tomcat-6.0.14/conf/web.xml ...DONE.

Adding SJS Tomcat Agent Filter and Form login authentication to selected Web
applications ...DONE
```

Implications of Specific Deployment Scenarios in Agent for Apache Tomcat 6.0

The following section refers to a specific deployment scenario involving Policy Agent 2.2 for Apache Tomcat 6.0.

Installing a J2EE Agent on Multiple Apache Tomcat 6.0 Instances

Once a J2EE agent is installed for a particular Apache Tomcat 6.0 instance, you can install the agent on another instance on the same machine by running the `agentadmin --install` command. Once prompted to enter the appropriate server instance name, enter the server configuration directory and unique instance name that will enable the agent to distinguish the first instance from consecutive instances.

Summary of a J2EE Agent Installation in Policy Agent 2.2

At the end of the installation process, the installation program prints the status of the installation along with the installed J2EE agent information. The information that the program displays can be very useful. For example, the program displays the agent instance name, which is needed when configuring a remote instance. The program also displays the location of specific files, which can be of great importance. In fact, you might want to view the installation log file once the installation is complete, before performing the post-installation steps as described in Chapter 4, "Post-Installation Tasks of Policy Agent 2.2 for Apache Tomcat 6.0."

The location of directories displayed by the installer are specific. However, throughout this guide and specifically in Summary of Agent Installation shown in this section, *PolicyAgent-base* is used to describe the directory where the distribution files are stored for a specific J2EE agent.

The following example serves as a quick description of the location of the J2EE agent base directory (*PolicyAgent-base*) of Policy Agent 2.2 for Apache Tomcat 6.0.

EXAMPLE 3–1 Policy Agent Base Directory of Agent for Apache Tomcat 6.0

The following directory represents *PolicyAgent-base* of Agent for Apache Tomcat 6.0:

Agent-HomeDirectory`/j2ee_agents/am_tomcat_agent`

where *Agent-HomeDirectory* is the directory you choose in which to unpack the J2EE agent binaries.

Information regarding the location of the J2EE agent base directory is explained in detail in "Location of the J2EE Agent Base Directory in Policy Agent 2.2" on page 46.

The following type of information is printed by the installer:

```
SUMMARY OF AGENT INSTALLATION
-----------------------------
Agent instance name: Agent_001
Agent Configuration file location:
PolicyAgent-base/Agent_001/config/AMAgent.properties
Agent Audit directory location:
PolicyAgent-base/Agent_001/logs/audit
Agent Debug directory location:
PolicyAgent-base/Agent_001/logs/debug

Install log file location:
PolicyAgent-base/logs/audit/install.log

Thank you for using Access Manager Policy Agent
```

Once the agent is installed, the directories shown in the preceding example are created in the `agent_00x` directory, which for this example is specifically `Agent_001`. Those directories and files are briefly described in the following paragraphs.

PolicyAgent-base`/Agent_001/config/AMAgent.properties`

Location of the J2EE agent `AMAgent.properties` configuration file for the agent instance. Every instance of a J2EE agent has a unique copy of this file. You can configure this file to meet your site's requirements. For more information, see the following sections:

- Appendix B, "J2EE Agent `AMAgent.properties` Configuration File in Policy Agent 2.2"
- "Key Features and Tasks Performed With the J2EE `AMAgent.properties` Configuration File" on page 81

PolicyAgent-base`/Agent_001/logs/audit`

Location of the J2EE agent local audit trail.

PolicyAgent-base`/Agent_001/logs/debug`

Location of all debug files required to debug an agent installation or configuration issue.

PolicyAgent-base`/logs/audit/install.log`
: Location of the file that has the agent install file location. If the installation failed for any reason, you can look at this file to diagnose the issue.

◆ ◆ ◆ CHAPTER 4

Post-Installation Tasks of Policy Agent 2.2 for Apache Tomcat 6.0

This chapter provides information about configuration and other post-installation considerations and tasks as follows:

- "Common Post-Installation Steps for All J2EE Agents in Policy Agent 2.2" on page 69
- "Post-Installation Steps Specific to Agent for Apache Tomcat 6.0" on page 71
- "Conditional Post-Installation Steps for J2EE Agents in Policy Agent 2.2" on page 73

After completing the applicable tasks described in this chapter, perform the tasks to configure the agent to your site's specific needs as explained in Chapter 5, "Managing Policy Agent 2.2 for Apache Tomcat 6.0."

Common Post-Installation Steps for All J2EE Agents in Policy Agent 2.2

The tasks described in this section apply to all J2EE agent installations.

Updating the Agent Profile for J2EE Agents in Policy Agent 2.2

This procedure is not required. The agent profile is created and updated in Access Manager Console. The agent profile should originally be created prior to installing an agent. However, after you install a J2EE agent, you can update the agent profile at anytime. If you do update the agent profile in Access Manager Console, you must then configure the J2EE agent accordingly as described in this section.

▼ To Update the Agent Profile for J2EE Agents in Policy Agent 2.2

Before You Begin Change the agent profile in Access Manager using Access Manager Console. For more information about the agent profile, see "Creating a J2EE Agent Profile" on page 52.

1 **Change the password in the password file to match the new password you just created in Access Manager Console as a part of the agent profile.**

The password file should originally have been created as a J2EE agent pre-installation task. For more information about pre-installation, see"Preparing to Install Agent for Apache Tomcat 6.0" on page 57.

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

For more information on this command, see "`agentadmin --encrypt`" on page 41.

3 **Access the J2EE agent** `AMAgent.properties` **configuration file at the following location:**

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

4 **In this configuration file, edit the property for the agent ID to match the new ID in the agent profile as follows:**

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

where *agentID* represents the new agent ID that you created for the agent profile in Access Manager Console.

5 **Edit the property for the agent password as follows:**

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

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

6 **Restart the J2EE agent container.**

The container needs to be restarted because neither property that you edited in this task is hot-swap enabled.

Deploying the Agent Application for J2EE Agents in Policy Agent 2.2

The task described in this section is required. Deploy the URI for the agent application using the deployment container. The agent application is a housekeeping application used by the agent for notifications and other internal functionality. This application is bundled with the agent binaries and can be found at the following location:

PolicyAgent-base/`etc/agentapp.`*extension*

where *extension* refers to the `.war` extension or the `.ear` extension. The extension varies depending on the deployment container.

For more information about the Policy Agent base directory (*PolicyAgent-base*), see "J2EE Agent Directory Structure in Policy Agent 2.2" on page 46.

The `agentapp` application has to be deployed as a post installation step. In order for the agent to function correctly, this application must be deployed on the agent-protected deployment container instance using the same URI that was supplied during the agent installation process (optionally, you can add a hyper link to and from the relevant prompt). For example during the installation process, if you entered `/agentapp` as the deployment URI for the agent application, then use that same context path to deploy the `.war` or `.ear` file in the deployment container.

Using the administration console or command-line utilities of your deployment container, deploy this application using Application Context Path as the URI specified during agent installation.

Post-Installation Steps Specific to Agent for Apache Tomcat 6.0

Once you have installed Policy Agent 2.2 for Apache Tomcat 6.0 and you have performed the post-installation steps that apply to all J2EE agents in the Policy Agent 2.2 release, complete the following agent-specific steps.

Installing the Agent Filter for the Deployed Application on Agent for Apache Tomcat 6.0

The agent filter can be installed by modifying the deployment descriptor of the application that needs to be protected.

▼ To Install the Agent Filter for the Deployed Application on Agent for Apache Tomcat 6.0

Note – By default, only the Manager web application and the administration web application are protected when the agent filter is in the J2EE_POLICY mode.

The following steps explain how to install the agent filter for the application you want the agent to protect:

1 **To install the agent filter, ensure that the application is not currently deployed on Apache Tomcat 6.0.**

If it is currently deployed, remove it before proceeding any further.

2 **Create the necessary backup files for the deployed application's deployment descriptors.**

Since you will modify the deployment descriptor in the next step, creating backup files at this point is important.

3 (Conditional) If the agent filter is not deployed in the global deployment descriptor, modify the deployed application's deployment descriptor by editing the application's `web.xml` descriptor.

a. Add the `<filter>` element by adding the following lines:

```
<filter>
<filter-name>Agent</filter-name>
<display-name>Agent</display-name>
<description>Identity Server Policy Agent Filter</description>
<filter-class>com.sun.identity.agents.filter.AmAgentFilter</filter-class>
</filter>
```

b. Add the `<filter-mapping>` element by adding the following lines:

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

Next Steps If you want to protect your application with J2EE declarative security, you must first perform the tasks described in "Configuring J2EE Declarative Security for Apache Tomcat 6.0 Related Web Applications" on page 73. You can also access the sample application in the *PolicyAgentBase*/`sampleapp` directory to learn how to build and deploy an application. The `sampleapp` directory is by no means a full fledged J2EE application. Rather it is a simple application that provides you with a quick reference to application specific deployment descriptors and various deployment modes of a J2EE agent. Once you successfully deploy the `sampleapp` and test all of its features, you can use it as a reference to other applications that will be protected by the J2EE agent.

Once the `web.xml` deployment descriptor is modified to reflect the new `<DOCTYPE>` and `<filter>` elements, the agent filter is added to the application. You can now redeploy your application on Apache Tomcat 6.0.

Note – Ensure that role-to-principal mappings in container specific deployment descriptors are replaced with Access Manager roles or principals. You can retrieve Access Manager roles or principals for Access Manager 7 by issuing the `agentadmin --getUuid` command. For more information on the `agentadmin --getUuid` command, see "`agentadmin --getUuid`" on page 43.

You can also retrieve the universal ID for the user (UUID) using Access Manager 7 Console to browse the user profile.

Conditional Post-Installation Steps for J2EE Agents in Policy Agent 2.2

Steps described in this section might be required, depending on your site's specific deployment.

Configuring J2EE Declarative Security for Apache Tomcat 6.0 Related Web Applications

The role-to-principal mappings in Apache Tomcat 6.0 deployment descriptors must be replaced with Access Manager roles or principals. The tasks described in this section include steps for changing the deployment descriptors of the Manager web application, the administration web application, and the host manager web application, thereby configuring J2EE declarative security for these applications.

- "To Create and Assign Access Manager Roles" on page 74
- "To Allow Access Manager Users to Access the Manager Web Application" on page 74
- "To Allow Access Manager Users to Access the Administration Web Application" on page 76
- "To Allow Access Manager Users to Access the Host Manager Web Application" on page 78

By default, Agent for Apache Tomcat 6.0 protects the Manager web application, the administration web application, and the host manager web application with J2EE security. This default configuration is established by the J2EE agent installer, which sets the agent filter mode to J2EE_POLICY in the J2EE agent `AMAgent.properties` configuration file as follows:

```
com.sun.identity.agents.config.Filter.mode = J2EE_POLICY
```

To protect the Manager web application, the administration web application, and the host manager web application with a filter mode other than J2EE_POLICY, change or add to the preceding setting accordingly in order to change the filter mode for these applications to URL_POLICY mode or ALL mode. The following example demonstrates these three applications set to specific filter modes. The administration web application is set to URL_POLICY mode while the Manager web application and the host manager web application are set to ALL mode.

```
com.sun.identity.agents.config.Filter.mode[admin] = URL_POLICY
com.sun.identity.agents.config.Filter.mode[manager] = ALL
com.sun.identity.agents.config.Filter.mode[host-manager] = ALL
```

After you have set the filter mode for each of these applications to the mode that best suits your site's deployment, perform the steps detailed in the following task descriptions.

▼ To Create and Assign Access Manager Roles

Using Access Manager Console, create Administrator users and Manager users as outlined in this task. For detailed information about Access Manager users and roles, see *Sun Java System Access Manager 7 2005Q4 Administration Guide*.

1. **Create the following roles: an administer role named `admin` and a manager role named `manager`.**

   The tasks that follow explain how to edit the appropriate `web.xml` files in Apache Tomcat 6.0 to add these role names. If the role names in Access Manager do not match role names in the respective `web.xml` files, the result is that access is denied to the respective Apache Tomcat 6.0 application.

2. **Create and assign users to the newly created roles.**

   Users assigned to the `admin` role can log in to the administration web application and the host manager web application. Users assigned the `manager` role can log in to the Manager web application.

▼ To Allow Access Manager Users to Access the Manager Web Application

Using the Apache Tomcat 6.0 instance, add the appropriate users and roles to the Manager web application's `web.xml` file as described in this task. The method for adding users to the `web.xml` file is not the same for Access Manager 7 and Access Manager 6.3. The differences relate to how user and role information is retrieved. Access Manager 7 takes advantage of a universal ID (UUID) system of identification while Access Manager 6.3 uses the distinguished name (DN) of users. Universal ID retrieval is achieved with the `agentadmin` program. For more information about the specific `agentadmin` commands to use, see "`agentadmin --getUuid`" on page 43.

1. **Change to the following directory:**

   ```
   $CATALINA_HOME/server/webapps/manager/WEB-INF
   ```

2. **Open the `web.xml` file.**

3. **Retrieve user and role information for the Manager role using the appropriate method according to the version of Access Manager you are configuring as follows:**

Access Manager 7	Use Universal ID for identification information.
Access Manager 6.3 Patch 1 or Greater	Use DN for identification information.

4. **Delete the Manager security role.**

   This role is defined in the `<role-name>` element under the `<security-role>` element.

5 **Create a new Manager security role using the user and role information created previously in Access Manager as described in "To Create and Assign Access Manager Roles" on page 74.**

The following examples demonstrate how to create a new Manager security role for Access Manager 7 and Access Manager 6.3 Patch 1 or greater.

- **Security Role Element for Access Manager 7**

  For this example, the following values apply to the universal ID for the Manager role in Access Manager 7, where *realmName* is a representation of organization name:

userName	`id=manager`
IdType	`ou=role`
realmName	`dc=subexample,dc=example,dc=com`

  The preceding values are used in the following example of a universal ID for the Manager role in Access Manager 7:

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

  The following is an example of a security role element, given the preceding universal ID information for the Manager role in Access Manager 7:

  ```
  <security-role>
  <role-name>id=manager,ou=role,dc=subexample,dc=example,dc=com</role-name>
  </security-role>
  ```

- **Security Role Element for Access Manager 6.3 Patch 1 or Greater**

  The following is an example of a role DN for the Manager role in Access Manager 6.3 where the organization is represented by `dc=subexample,dc=example,dc=com`:

  ```
  cn=manager,ou=groups,dc=subexample,dc=example,dc=com
  ```

  The following is an example of a security role element, given the preceding DN information for the Manager role in Access Manager 6.3:

  ```
  <security-role>
  <role-name>cn=manager,ou=groups,dc=subexample,dc=
  example,dc=com</role-name></security-role>
  ```

6 **Replace the Manager role defined in the `<role-name>` element under the `<auth-constraint>` element.**

This Manager role should be replaced with the contents of the `<role-name>` element as described in the previous step and demonstrated as follows:

- **Manager Role for Access Manager 7**

After the Manager role definition has been replaced, the `<auth-constraint>` element for the Manager role in Access Manager 7 for the `dc=subexample,dc=example,dc=com` realm would appear as such:

```
<auth-constraint>
<role-name>id=manager,ou=role,dc=subexample,dc=example,dc=com</role-name>
</auth-constraint>
```

- **Manager Role for Access Manager 6.3 Patch 1 or Greater**

  After the Manager role definition has been replaced, the `<auth-constraint>` element for the Manager role in Access Manager 6.3 for the `dc=subexample,dc=example,dc=com` organization would appear as such:

```
<auth-constraint>
<role-name>cn=manager,ou=groups,dc=subexample,dc=example,dc=com</role-name>
</auth-constraint>
```

▼ To Allow Access Manager Users to Access the Administration Web Application

In the Apache Tomcat 6.0 instance, add the appropriate users and roles to the administration web application's `web.xml` file as described in this task. This task is similar to the preceding task in that the two tasks both apply to Access Manager 6.3 Patch 1 or greater and Access Manager 7. Use the information in this task that applies to your site's deployment.

1 **Change to the following directory:**

```
$CATALINA_HOME/server/webapps/admin/WEB-INF
```

2 **Open the `web.xml` file.**

3 **Retrieve user and role information for the Administrator role using the appropriate method according to the version of Access Manager you are configuring:**

Access Manager 7	Use Universal ID for identification information.
Access Manager 6.3 Patch 1 or Greater	Use DN for identification information.

4 **Delete the Administrator security role.**

This role is defined in the `<role-name>` element under the `<security-role>` element.

5 **Create a new Administrator security role using the user and role information created previously in Access Manager as described in "To Create and Assign Access Manager Roles" on page 74.**

The following examples demonstrate how to create a new Administrator security role for Access Manager 7 and Access Manager 6.3 Patch 1 or greater.

- **Security Role Element for Access Manager 7**

  For this example, the following values apply to the universal ID for the Administrator role in Access Manager 7, where *realmName* is a representation of organization name:

userName	`id=admin`
IdType	`ou=role`
realmName	`dc=subexample,dc=example,dc=com`

  The preceding values are used in the following example of a universal ID for the Administrator role in Access Manager 7:

  ```
  id=admin,ou=role,dc=subexample,dc=example,dc=com
  ```

  The following is an example of a security role element, given the preceding universal ID information for the Administrator role in Access Manager 7:

  ```
  <security-role>
  <role-name>id=admin,ou=role,dc=subexample,dc=example,dc=com</role-name>
  </security-role>
  ```

- **Security Role Element for Access Manager 6.3 Patch 1 or Greater**

  The following is an example of a role DN for the Administrator role in Access Manager 6.3 where the organization is represented by `dc=subexample,dc=example,dc=com`:

  ```
  cn=admin,ou=groups,dc=subexample,dc=example,dc=com
  ```

  The following is an example of a security role element given the preceding DN information for the Administrator role in Access Manager 6.3:

  ```
  <security-role>
  <role-name>cn=admin,ou=groups,dc=subexample,dc=
  example,dc=com</role-name></security-role>
  ```

6 **Replace the Administrator role defined in the `<role-name>` element under the `<auth-constraint>` element.**

This Administrator role should be replaced with the contents of the `<role-name>` element as described in the previous step and demonstrated as follows:

- **Administrator Role for Access Manager 7**

After the Administrator role definition has been replaced, the `<auth-constraint>` element for the Administrator role in Access Manager 7 for the `dc=subexample,dc=example,dc=com` realm would appear as such:

```
<auth-constraint>
<role-name>id=admin,ou=role,dc=subexample,dc=example,dc=com</role-name>
</auth-constraint>
```

- **Administrator Role for Access Manager 6.3 Patch 1 or Greater**

  After the Administrator role definition has been replaced, the `<auth-constraint>` element for the Administrator role in Access Manager 6.3 for the `dc=subexample,dc=example,dc=com` organization would appear as such:

  ```
  <auth-constraint>
  <role-name>cn=admin,ou=groups,dc=subexample,dc=example,dc=com</role-name>
  </auth-constraint>
  ```

▼ To Allow Access Manager Users to Access the Host Manager Web Application

● **Follow similar steps to those described in "To Allow Access Manager Users to Access the Administration Web Application" on page 76 by editing the `web.xml` file of the host manager web application at the following location:**

`$CATALINA_HOME/server/webapps/host-manager/WEB-INF`

All the remaining steps for configuring the host manager web application with declarative security are the same as for the administration web application.

Since both applications are accessible by users assigned to the role `admin`, the `web.xml` files for both applications must be edited in the same manner.

Creating the Necessary URL Policies

If the agent is installed and configured to operate in the URL_POLICY mode or ALL mode, the appropriate URL policies must be created. For instance, if Apache Tomcat 6.0 is available on port 8080 using HTTP protocol, at least a policy must be created to allow access to the following resource:

```
http://myhost.mydomain.com:8080/sampleApp/
```

where `sampleApp` is the context URI for the sample application.

If no policies are defined and the agent is configured to operate in the URL_POLICY mode or ALL mode, then no user is allowed access to Apache Tomcat 6.0 resources. See *Sun Java System Access Manager 7 2005Q4 Administration Guide* to learn how to create these policies using the Access Manager Console or command-line utilities.

◆ ◆ ◆ CHAPTER 5

Managing Policy Agent 2.2 for Apache Tomcat 6.0

After installing Policy Agent 2.2 for Apache Tomcat 6.0 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 the agent accordingly.

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:

- "Key Features and Tasks Performed With the J2EE `AMAgent.properties` Configuration File" on page 81
- "Key Features and Tasks Performed With the J2EE `agentadmin` Program" on page 105
- "Key Features and Tasks Performed With the J2EE Agent API" on page 106

Key Features and Tasks Performed With the J2EE `AMAgent.properties` Configuration File

The J2EE agent `AMAgent.properties` configuration file is a text file of configuration properties that you can modify to change J2EE agent behavior.

Caution – The content of the J2EE agent `AMAgent.properties` configuration file is very sensitive. Changes made can result in changes in how the agent works. Errors made can cause the agent to malfunction.

This section describes the most important details of the J2EE agent `AMAgent.properties` configuration file, such as how specific properties can be modified to produce specific results. The topics described are typically those of greatest interest in real-world deployment scenarios. This section does not cover every property in the file. For a list and description of every property, see Appendix B, "J2EE Agent `AMAgent.properties` Configuration File in Policy Agent 2.2."

The following is the location of the `AMAgent.properties` file;

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

For more information about the Policy Agent 2.2 directory structure, see "J2EE Agent Directory Structure in Policy Agent 2.2" on page 46.

- "Hot-Swap Mechanism in J2EE Agents" on page 82
- "List Constructs in the J2EE `AMAgent.properties` Configuration File" on page 83
- "Map Constructs in the J2EE `AMAgent.properties` Configuration File" on page 84
- "J2EE Property Configuration: Application Specific or Global" on page 85
- "J2EE Agent Filter Modes" on page 86
- "Enabling Web-Tier Declarative Security in J2EE Agents" on page 88
- "Enabling Failover in J2EE Agents" on page 93
- "Login Attempt Limit in J2EE Agents" on page 95
- "Redirect Attempt Limit in J2EE Agents" on page 96
- "Not-Enforced URI List in J2EE Agents" on page 96
- "Fetching Attributes in J2EE Agents" on page 97
- "Configuring FQDN Handling in J2EE Agents" on page 102
- "Using Cookie Reset Functionality in J2EE Agents" on page 103
- "Enabling Port Check Functionality in J2EE Agents" on page 104

Hot-Swap Mechanism in J2EE Agents

Certain property keys in the J2EE agent `AMAgent.properties` configuration file 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 Access Manager 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 configuration changes made in the J2EE agent `AMAgent.properties` configuration file 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.

List Constructs in the J2EE `AMAgent.properties` Configuration File

Certain property keys in the J2EE agent `AMAgent.properties` configuration file are specified as lists. A list construct has the following format:

```
<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 5–1 Example of List Constructs in J2EE `AMAgent.properties` File

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

Map Constructs in the J2EE `AMAgent.properties` Configuration File

Certain property keys in the J2EE agent are specified as maps. A map construct has the following format:

```
<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 5–2 Example of Map Constructs in J2EE AMAgent.properties File

```
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 property keys in the J2EE agent `AMAgent.properties` configuration file 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. An application specific property has the following format:

```
<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 5–3 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 J2EE agent `AMAgent.properties` configuration file 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 following configuration property is used to control the mode of the agent filter:

```
com.sun.identity.agents.config.filter.mode
```

The value for this 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.

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.

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 Access Manager 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 variousAccess Manager 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 Access Manager 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 88 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 Access Manager.

J2EE Agent Filter Mode - URL_POLICY

In this mode, the agent filter is used to enforce various URL policies that may be defined in Access Manager. 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 Access Manager. 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 in the J2EE agent `AMAgent.properties` configuration file 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 property in the J2EE agent `AMAgent.properties` configuration file:**

`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 following property in the J2EE agent `AMAgent.properties` configuration file as shown:

```
com.sun.identity.agents.config.login.form[0] = /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 in the J2EE agent `AMAgent.properties` configuration file. 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 J2EE agent `AMAgent.properties` configuration file. For example, other entries could be:

```
com.sun.identity.agents.config.login.error.uri[1] = /BankApp/SignOn
```

```
com.sun.identity.agents.config.login.error.uri[2] = /ERP/LoginServlet
```

Please 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 in the J2EE agent `AMAgent.properties` configuration file, 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 91.
- 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 90.

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

The J2EE agent AMAgent.properties configuration file allows 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 Access Manager login page.

Using the J2EE agent `AMAgent.properties` configuration file, 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 using the property `com.sun.identity.agents.config.login.content.file` in the J2EE agent `AMAgent.properties` configuration 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 setting the value of the configuration property as follows:`com.sun.identity.agents.config.login.use.internal` as `false`.

   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 Access Manager instance for which the agent has been configured becomes unavailable, the agent will switch to the next Access Manager instance as specified in the J2EE agent `AMAgent.properties` configuration file. This setup can be achieved by implementing the following steps.

▼ To Enable Failover in J2EE Agents

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

Configure the following property to create the list:

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

You may specify more than one login URL as follows:

`com.sun.identity.agents.config.login.url[0]` = *primary-AM-server*

`com.sun.identity.agents.config.login.url[1]` = *failover-AM-server1*

`com.sun.identity.agents.config.login.url[2]` = *failover-AM-server2*

primary-AM-server	Represents the URL of the primary Access Manager instance to which users are redirected for authentication.
failover-AM-server1	Represents the URL of the Access Manager instance to which users are redirected for authentication if the primary Access Manager instance fails.
failover-AM-server2	Represents the URL of the Access Manager instance to which users are redirected for authentication if the primary Access Manager instance fails and the first failover Access Manager instance fails.

If a URL list is provided to this property, `com.sun.identity.agents.config.login.url`, the agent first tries to establish a connection to the first server (*primary-AM-server*) specified in the URL list. If the agent is successful in establishing this connection, it redirects the user to the Access Manager instance for authentication.

2 (Optional) Turn prioritization on for the failover lists by setting the following property to `true`:

`com.sun.identity.agents.config.login.url.prioritized`

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

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

Case 1: `com.sun.identity.agents.config.login.url.prioritized = true`

A value of `true` means that priority is established for the login URL list described in Step 1. The list was created by configuring the following property:

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

Therefore, the first URL on the list, which is abbreviated here as `.url[0]`, has a higher priority than `.url[1]` and `.url[1]` has higher priority than `.url[2]` and so on. If the server (*primary-AM-server*) specified in this example as the value for `.url[0]` 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*) associated with `.url[1]`. 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 Access Manager instance among the Access Manager instances that are running at the point in time the agent must redirect requests to an Access Manager instance.

Case 2: `com.sun.identity.agents.config.login.url.prioritized = false`

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*) associated with `.url[0]`. If *primary-AM-server* goes down then all subsequent requests are sent to the server (*failover-AM-server1*) associated with `.url[1]`. 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*) associated with `.url[2]` 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 Access Manager Naming Service URLs that may be used by the agent to get access to the various other service URLs that may be needed to serve the logged on user.

This can be done by using the following property:

`com.iplanet.am.naming.url`

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 Access Manager instances for which you want to enable failover, provide a URL list of the remote Access Manager 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 Access Manager 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 in the J2EE agent `AMAgent.properties` configuration file.

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. This is configured using the Login Attempt Limit configuration property. The configuration property that controls this behavior is as follows:

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

If a non-zero positive value is specified for this property in the J2EE agent `AMAgent.properties` configuration file, 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 Access Manager 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

The J2EE agent `AMAgent.properties` configuration file allows you to specify a list of URIs that are treated as not-enforced. Access to these resources is always granted by the agent. The configuration property that controls this list is as follows:

```
com.sun.identity.agents.config.notenforced.uri
```

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

```
com.sun.identity.agents.config.notenforced.uri[0] = /images/*

com.sun.identity.agents.config.notenforced.uri[1] = /public/*.html

com.sun.identity.agents.config.notenforced.uri[2] = /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.

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 following property:

```
com.sun.identity.agents.config.notenforced.uri.invert
```

When you set the value to `true` for this property, it makes the entries specified in the not-enforced list as enforced entries 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. This 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 2.2 allows the following attribute types to be fetched using the corresponding property from the J2EE agent `AMAgent.properties` configuration file:

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 100.

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 properties in the J2EE agent `AMAgent.properties` configuration file that can influence the attribute fetching process, see "Common Attribute Fetch Processing Related Properties" on page 101.

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 103.

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 Access Manager 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 Access Manager 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 Access Manager 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 this scenario occurs, it can have a severe impact on the system since the agent now detects 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 2.2" on page 34.

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.

TABLE 5–1 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

TABLE 5–1 The `agentadmin` Program: Supported Options *(Continued)*

Option	Task Performed
`--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 Access Manager application program interfaces (API) that can be used to further enhance the security of your application. Besides the Access Manager 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 2.2 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 2.2" on page 108.

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 107.

  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 107.

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 2.2

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

EXAMPLE 5–4 Usage of New J2EE Agent API

- Web Tier Use Case:

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

- EJB Tier Use Case:

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

Caution – This public API can only retrieve the `SSOToken` object in EJB context if the value of the following property in the J2EE agent `AMAgent.properties` file is set to `true` as shown:

```
com.sun.identity.agents.config.user.principal = true
```

CHAPTER 6

Uninstalling Policy Agent 2.2 for Apache Tomcat 6.0

The `agentadmin` program is used for initiating the installation and uninstallation programs of Policy Agent 2.2 for Apache Tomcat 6.0. The difference is that the installation program is started with the `--install` option while the uninstallation program is started with the `--uninstall` option. For more information about the `agentadmin` program, see "Key Features and Tasks Performed With the J2EE `agentadmin` Program" on page 105. The uninstallation program is similar to the installation program in that it provides step by step explanations of the information you need to enter. However, the uninstallation program has fewer and simpler steps.

The uninstallation process follows a series of tasks similar to the installation process. First, perform the pre-uninstallation (preparation) steps. Then, perform the uninstallation, itself.

The first phase of uninstallation is the launching of the uninstallation program. The second phase of uninstallation involves interacting with the uninstallation program. During this phase, the program prompts you step by step to enter specific information while providing you with explanations about that information.

You must access the *PolicyAgent-base* directory for uninstallation-related tasks. For more information about this directory, see "J2EE Agent Directory Structure in Policy Agent 2.2" on page 46.

Preparing to Uninstall Agent for Apache Tomcat 6.0

Perform the pre-uninstallation (preparation) steps outlined in this section before uninstalling Policy Agent 2.2 for Apache Tomcat 6.0.

▼ To Prepare to Uninstall Agent for Apache Tomcat 6.0

To prepare for the uninstallation of Policy Agent 2.2 for Apache Tomcat 6.0, perform the following steps:

1 **Undeploy any protected applications from Apache Tomcat 6.0.**

Refer to the Apache Tomcat 6.0 documentation for more information.

2 **Restore the deployment descriptors of these applications to their original deployment descriptors.**

3 **Ensure that the Apache Tomcat 6.0 instance from which you are about to uninstall the agent is not running.**

4 **Undeploy the agent application.**

The agent application must be undeployed from Apache Tomcat 6.0 before the agent is uninstalled.

The agent application was installed during the post-installation steps. For more information about the installation of this application see Chapter 3, "Installing Policy Agent 2.2 for Apache Tomcat 6.0."

Uninstalling Agent for Apache Tomcat 6.0

This uninstallation process involves two phases as described in the following subsections.

Launching the Uninstallation Program of Agent for Apache Tomcat 6.0

Perform the steps outlined in this section to launch the uninstallation program of Policy Agent 2.2 for Apache Tomcat 6.0.

▼ To Launch the Uninstallation Program of Agent for Apache Tomcat 6.0

To launch the uninstallation program, perform the following steps:

1 **Change to the following directory:**

PolicyAgent-base`/bin`

This directory contains the `agentadmin` program, which is used for uninstalling a J2EE agent and for performing other tasks. For more information on the `agentadmin` program, see "Key Features and Tasks Performed With the J2EE `agentadmin` Program" on page 105.

2 **Issue one of the following commands:**

```
./agentadmin --uninstall
```

or

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

These two commands are different in that the `--uninstallAll` option removes *all* configured instances of the agent.

After you issue one of the preceding commands, the uninstallation program launches and presents you with the first prompt as illustrated in the following section.

Using the Uninstallation Program of Agent for Apache Tomcat 6.0

The steps in the uninstallation program are displayed in the following example. The interaction process of this uninstallation program is similar to that of the installation program. One difference is that the uninstallation program does not present a license agreement. For a more detailed explanation of the interaction process, see "Using the Installation Program of Agent for Apache Tomcat 6.0" on page 61.

Example of Uninstallation Program Interaction in Agent for Apache Tomcat 6.0

```
************************************************************************
Welcome to the Access Manager Policy Agent for Apache Tomcat 6.0 Servlet/JSP
Container

************************************************************************


Enter the complete path to the directory which is used by Tomcat Server to
store its configuration Files. This directory uniquely identifies the
Tomcat Server instance that is secured by this Agent.
[ ? : Help, ! : Exit ]
Enter the Tomcat Server Config Directory Path
[/opt/apache-tomcat-6.0.14/conf]:

-----------------------------------------------
SUMMARY OF YOUR RESPONSES
-----------------------------------------------
Tomcat Server Config Directory :
/opt/apache-tomcat-6.0.14/conf
Verify your settings above and decide from the choices below.
1. Continue with Uninstallation
2. Back to the last interaction
3. Start Over
4. Exit
```

```
Please make your selection [1]:
Removing the agent classpath from
/opt/apache-tomcat-6.0.14/bin/setclasspath.sh script ...DONE.

Deleting the config directory
Agent-HomeDirectory/j2ee_agents/tomcat_v6_agent/Agent_001/config
...DONE.

Removing SJS Tomcat Agent Realm from Server XML file :
/opt/apache-tomcat-6.0.14/conf/server.xml ...DONE.

Removing filter from Global deployment descriptor file :
/opt/apache-tomcat-6.0.14/conf/web.xml ...DONE.

Removing SJS Tomcat Agent Filter and Form login authentication from Web
applications ...DONE.


Uninstall log file location:
Agent-HomeDirectory/j2ee_agents/tomcat_v6_agent/logs/audit/uninstall.log

Thank you for using Access Manager Policy Agent
----------------
```

APPENDIX A

Silent Installation and Uninstallation of a J2EE Agent in Policy Agent 2.2

In addition to a standard installation and uninstallation of J2EE agents, you can perform a silent installation or uninstallation as described in this appendix.

About Silent Installation and Uninstallation of a J2EE Agent in Policy Agent 2.2

A silent installation or uninstallation refers to installing or uninstalling a program by implementing a script. The script is part of a state file. The script provides all the answers that you would normally supply to the installation or uninstallation program interactively. Running the script saves time and is useful when you want to install or uninstall multiple instances of Policy Agent using the same parameters in each instance.

Silent installation is a simple two-step process of generating a state file and then using that state file. To generate a state file, you record the installation or uninstallation process, entering all the required information that you would enter during a standard installation or uninstallation. Then you run the installation or uninstallation program with the state file as the input source.

Generating a State File for a J2EE Agent Installation

This section describes how to generate a state file for installing a J2EE agent. This task requires you to issue a command that records the information you will enter as you follow the agent installation steps. Enter all the necessary installation information in order to create a complete state file.

▼ To Generate a State File for a J2EE Agent Installation

To generate a state file for a J2EE agent installation , perform the following:

1 Change to the following directory:

PolicyAgent-base`/bin`

This directory contains the `agentadmin` program, which is used for installing a J2EE agent and for performing other tasks. For more information on the `agentadmin` program, see "Key Features and Tasks Performed With the J2EE `agentadmin` Program" on page 105.

2 Issue the following command:

```
./agentadmin --install --saveResponse filename
```

`-saveResponse`	An option that saves all of your responses to installation prompts in a state file.
filename	Represents the name that you choose for the state file.

3 Perform the installation as described in Chapter 3, "Installing Policy Agent 2.2 for Apache Tomcat 6.0"

Your answers to the prompts are recorded in the state file. When the installation is complete, the state file is created in the same directory where the installation program is located.

Note – When generated, a state file will have `read` permissions for all users. However, because the state file contains clear text passwords, it is recommended that you change the file permissions to restrict `read` and `write` access to the user `root`.

Using a State File for a J2EE Agent Silent Installation

The installation program does not validate inputs or the state in the silent installation. Ensure that the proper environment exists before performing a silent installation.

▼ To Install a J2EE Agent Using a State File

To perform a silent installation of a J2EE agent using a state file, perform the following:

1 Change to the following directory:

PolicyAgent-base`/bin`

At this point, this `bin` directory should contain the `agentadmin` program and the J2EE agent installation state file.

2 Issue the following command:

```
./agentadmin --install --useResponse filename
```

`-useResponse`	An option that directs the installer to run in non-interactive mode as it obtains all responses to prompts from the named state file.
filename	Represents the name of the state file from which the installer obtains all responses.

The installation takes place hidden from view. After completion, the program exits automatically and displays the prompt.

Generating a State File for a J2EE Agent Uninstallation

This section describes how to generate a state file for uninstalling a J2EE agent. This task requires you to issue a command that records the information you will enter as you follow the agent uninstallation steps. Enter all the necessary uninstallation information in order to create a complete state file.

▼ To Generate a State File for a J2EE Agent Uninstallation

To generate a state file for uninstallation of a J2EE agent, perform the following:

1 Change to the following directory:

PolicyAgent-base/`bin`

This directory contains the `agentadmin` program, which is used for uninstalling a J2EE agent and for performing other tasks. For more information on the `agentadmin` program, see "Key Features and Tasks Performed With the J2EE `agentadmin` Program" on page 105.

2 Issue the following command:

```
./agentadmin --uninstall --saveResponse filename
```

`-saveResponse`	An option that saves all of your responses to uninstallation prompts in a state file.
filename	Represents the name that you choose for the state file.

3 Perform the uninstallation as explained in Chapter 6, "Uninstalling Policy Agent 2.2 for Apache Tomcat 6.0."

Your answers to the prompts are recorded in the state file. When uninstallation is complete, the state file is created in the same directory where the uninstallation program is located.

Note – When generated, a state file will have `read` permissions for all users. However, because the state file contains clear text passwords, it is recommended that you change the file permissions to restrict `read` and `write`access to the user `root`.

Using a State File for a J2EE Agent Silent Uninstallation

The uninstallation program does not validate inputs or the state in the silent installation. Ensure that the proper environment exists before performing a silent uninstallation.

▼ To Uninstall a J2EE Agent Using a State File

To perform a silent uninstallation of a J2EE agent using a state file, perform the following:

1 **Change to the following directory:**

PolicyAgent-base/`bin`

At this point, this `bin` directory should contain the `agentadmin` program and the J2EE uninstallation state file.

2 **Issue the following command:**

```
./agentadmin --uninstall --useResponse filename
```

`-useResponse`	An option that runs the uninstallation process in non-interactive mode as all responses to prompts are obtained from the named state file.
filename	Represents the name of the state file from which the installer obtains all responses.

The uninstallation takes place hidden from view. After completion, the program exits automatically and displays the prompt.

APPENDIX B

J2EE Agent AMAgent.properties Configuration File in Policy Agent 2.2

The J2EE AMAgent.properties configuration file contains the necessary configuration properties needed for the agent to function properly. It also contains the necessary information needed for the Sun Java System Access Manager SDK to function properly in a client installation mode as used by the agent.

Caution – The content of the J2EE agent AMAgent.properties configuration file is very sensitive. Changes made can result in changes in how the agent works. Errors made can cause the agent to malfunction.

This appendix provides basic information about the J2EE AMAgent.properties configuration file. Specifically, this appendix describes where the configuration is located, provides a quick list of the properties, and provides the same list but with a simple description of each property. This appendix organizes the information as follows:

- "Location of the J2EE AMAgent.properties Configuration File" on page 118
- "List of Properties in the J2EE AMAgent.properties Configuration File" on page 118
- "Description of Properties in the J2EE AMAgent.properties Configuration File" on page 123

Each property is described in more detail in the actual J2EE AMAgent.properties configuration file. Furthermore, for an explanation of key features of this configuration file and tasks that you can accomplish with it, see "Key Features and Tasks Performed With the J2EE AMAgent.properties Configuration File" on page 81.

Location of the J2EE `AMAgent.properties` Configuration File

The following is the location of the J2EE `AMAgent.properties` configuration file:

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

For more information about the Policy Agent 2.2 directory structure, see "J2EE Agent Directory Structure in Policy Agent 2.2" on page 46.

List of Properties in the J2EE `AMAgent.properties` Configuration File

This section provides a list of all the J2EE agent properties in the `AMAgent.properties` configuration file. The properties are divided into categories according to the aspect of Policy Agent that each property enables you to modify.

Filter Operation Mode Property
`com.sun.identity.agents.config.filter.mode`

User Mapping Properties
`com.sun.identity.agents.config.user.mapping.mode[]`
`com.sun.identity.agents.config.user.attribute.name`
`com.sun.identity.agents.config.user.principal`
`com.sun.identity.agents.config.user.token`

Client Identification Properties
`com.sun.identity.agents.config.client.ip.header`
`com.sun.identity.agents.config.client.hostname.header`

Configuration Reload Interval Property
`com.sun.identity.agents.config.load.interval`

Local Identification Properties
`com.sun.identity.agents.config.locale.language`
`com.sun.identity.agents.config.locale.country`

Organization Name Property
`com.sun.identity.agents.config.organization.name`

Audit Log Properties

```
com.sun.identity.agents.config.audit.accesstype
com.sun.identity.agents.config.log.disposition
com.sun.identity.agents.config.remote.logfile
com.sun.identity.agents.config.local.logfile
com.sun.identity.agents.config.local.log.rotate
com.sun.identity.agents.config.local.log.size
```

Web Service Processing Properties

```
com.sun.identity.agents.config.webservice.enable
com.sun.identity.agents.config.webservice.endpoint[]
com.sun.identity.agents.config.webservice.process.get.enable
com.sun.identity.agents.config.webservice.authenticator
com.sun.identity.agents.config.webservice.internalerror.content
com.sun.identity.agents.config.webservice.autherror.content
```

Access Denied URI Property

```
com.sun.identity.agents.config.access.denied.uri
```

Form Login Processing Properties

```
com.sun.identity.agents.config.login.form[]
com.sun.identity.agents.config.login.error.uri[]
com.sun.identity.agents.config.login.use.internal
com.sun.identity.agents.config.login.content.file
```

Local Authentication Processing Properties

```
com.sun.identity.agents.config.auth.handler[]
com.sun.identity.agents.config.logout.handler[]
com.sun.identity.agents.config.verification.handler[]
```

Goto Parameter Name Property

```
com.sun.identity.agents.config.redirect.param
```

Login URL Property

```
com.sun.identity.agents.config.login.url[]
```

Login URL Prioritized Flag Property

```
com.sun.identity.agents.config.login.url.prioritized
```

Agent Server Properties

```
com.sun.identity.agents.config.agent.host
com.sun.identity.agents.config.agent.port
com.sun.identity.agents.config.agent.protocol
```

Login Attempt Limit Property

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

URL Decode SSO Token Property

```
com.sun.identity.agents.config.sso.decode
```

SSO Cache Enable Property

```
com.sun.identity.agents.config.amsso.cache.enable
```

Cookie Reset Processing Properties

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

CDSSO Processing Properties

```
com.sun.identity.agents.config.cdsso.enable
com.sun.identity.agents.config.cdsso.redirect.uri
com.sun.identity.agents.config.cdsso.cdcservlet.url[]
com.sun.identity.agents.config.cdsso.clock.skew
com.sun.identity.agents.config.cdsso.trusted.id.provider[]
```

Logout Processing Properties

```
com.sun.identity.agents.config.logout.application.handler[]
com.sun.identity.agents.config.logout.uri[]
com.sun.identity.agents.config.logout.request.param[]
com.sun.identity.agents.config.logout.introspect.enabled
com.sun.identity.agents.config.logout.entry.uri[]
```

FQDN Processing Properties

```
com.sun.identity.agents.config.fqdn.check.enable
com.sun.identity.agents.config.fqdn.default
com.sun.identity.agents.config.fqdn.mapping[]
```

Legacy User Agent Processing Properties

```
com.sun.identity.agents.config.legacy.support.enable
com.sun.identity.agents.config.legacy.user.agent[]
com.sun.identity.agents.config.legacy.redirect.uri
```

Custom Response Headers Property

```
com.sun.identity.agents.config.response.header[]
```

Redirect Attempt Limit Property

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

Port Check Processing Properties

```
com.sun.identity.agents.config.port.check.enable
com.sun.identity.agents.config.port.check.file
com.sun.identity.agents.config.port.check.setting[]
```

Not-Enforced URI Processing Properties

```
com.sun.identity.agents.config.notenforced.uri[]
com.sun.identity.agents.config.notenforced.uri.invert
com.sun.identity.agents.config.notenforced.uri.cache.enable
com.sun.identity.agents.config.notenforced.uri.cache.size
```

Not-Enforced Client IP Processing Properties

```
com.sun.identity.agents.config.notenforced.ip[]
com.sun.identity.agents.config.notenforced.ip.invert
com.sun.identity.agents.config.notenforced.ip.cache.enable
com.sun.identity.agents.config.notenforced.ip.cache.size
```

Common Attribute Fetch Processing Properties

```
com.sun.identity.agents.config.attribute.cookie.separator
com.sun.identity.agents.config.attribute.date.format
com.sun.identity.agents.config.attribute.cookie.encode
```

Profile Attribute Processing Properties

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

Session Attribute Processing Properties

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

Response Attribute Processing Properties

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

Bypass Principal List Property

`com.sun.identity.agents.config.bypass.principal[]`

Privileged Attribute Processing Properties

`com.sun.identity.agents.config.default.privileged.attribute[]`
`com.sun.identity.agents.config.privileged.attribute.type[]`
`com.sun.identity.agents.config.privileged.attribute.tolowercase[]`
`com.sun.identity.agents.config.privileged.session.attribute[]`

Service Resolver Property

`com.sun.identity.agents.config.service.resolver`

Agent Username and Password Properties

`com.sun.identity.agents.app.username`
`com.iplanet.am.service.secret`

Encryption Key Properties

`am.encryption.pwd`
`com.sun.identity.client.encryptionKey`

Debug Service Properties

`com.iplanet.services.debug.level`
`com.iplanet.services.debug.directory`

SSO Token Cookie Name Property

`com.iplanet.am.cookie.name`

Naming Service URL Property

`com.iplanet.am.naming.url`

Session Client Properties

`com.iplanet.am.notification.url`
`com.iplanet.am.session.client.polling.enable`
`com.iplanet.am.session.client.polling.period`

Encryption Provider Property

`com.iplanet.security.encryptor`

User Data Cache Update Time Property

`com.iplanet.am.sdk.remote.pollingTime`

Service Data Cache Update Time Property

`com.sun.identity.sm.cacheTime`

SAML Service Properties

```
com.iplanet.am.localserver.protocol
com.iplanet.am.localserver.host
com.iplanet.am.localserver.port
```

Authentication Service Properties

```
com.iplanet.am.server.protocol
com.iplanet.am.server.host
com.iplanet.am.server.port
```

Policy Client Properties

```
com.sun.identity.agents.server.log.file.name
com.sun.identity.agents.logging.level
com.sun.identity.agents.notification.enabled
com.sun.identity.agents.notification.url
com.sun.identity.agents.polling.interval
com.sun.identity.policy.client.cacheMode
com.sun.identity.policy.client.booleanActionValues
com.sun.identity.policy.client.resourceComparators
com.sun.identity.policy.client.clockSkew
```

Description of Properties in the J2EE `AMAgent.properties` Configuration File

This section provides a brief description of all the J2EE agent properties in the `AMAgent.properties` configuration file. The properties are divided into categories according to the aspect of Policy Agent that each property enables you to modify.

Filter Operation Mode Property

- **`com.sun.identity.agents.config.filter.mode`**

Hot-swap enabled: No

This property specifies the mode of operation of the filter. The following are valid values for this property:

```
NONE
SSO_ONLY
URL_POLICY
J2EE_POLICY
ALL
```

This property can also be specified as an application specific property. However, the global property must be overwritten.

User Mapping Properties

```
com.sun.identity.agents.config.user.mapping.mode[]
com.sun.identity.agents.config.user.attribute.name
com.sun.identity.agents.config.user.principal
com.sun.identity.agents.config.user.token
```

- **`com.sun.identity.agents.config.user.mapping.mode[]`**

Hot-swap enabled: No

This property specifies the mechanism by which the user ID used on the protected server for the authenticated user is determined by the J2EE agent. The following are valid values for this property:

```
USER_ID
PROFILE_ATTRIBUTE
HTTP_HEADER
SESSION_PROPERTY
```

- **`com.sun.identity.agents.config.user.attribute.name`**

Hot-swap enabled: No

This property specifies the name of the profile attribute, HTTP header, or session property that contains the user ID used on the protected server for the authenticated user.

Key Properties Affecting This Property

This property is *not* used when the following property is set as shown:

```
com.sun.identity.agents.config.user.mapping.mode = USER_ID
```

- **`com.sun.identity.agents.config.user.principal`**

Hot-swap enabled: No

This property is a flag that indicates how the user is authenticated on the protected server. When this property is set to `true`, the principal of the authenticated user, not simply the user ID, is used for authentication purposes.

Key Properties Affecting This Property

This property is only used when the following property is set as shown:

```
com.sun.identity.agents.config.user.mapping.mode = USER_ID
```

- **`com.sun.identity.agents.config.user.token`**

Hot-swap enabled: No

This property specifies a session property name which contains the user ID of the authenticated user in session.

Key Properties Affecting This Property

This property is only used when the following properties are set as shown:

```
com.sun.identity.agents.config.user.mapping.mode = USER_ID
com.sun.identity.agents.config.user.principal = false
```

Client Identification Properties

`com.sun.identity.agents.config.client.ip.header`
`com.sun.identity.agents.config.client.hostname.header`

- **`com.sun.identity.agents.config.client.ip.header`**

Hot-swap enabled: No

This property specifies an HTTP header name that holds the IP address of the client. If you will not employ this property, leave it blank.

- **`com.sun.identity.agents.config.client.hostname.header`**

Hot-swap enabled: No

This property specifies an HTTP header name that holds the hostname of the client. If you do not use this property, leave it blank.

Configuration Reload Interval Property

- **`com.sun.identity.agents.config.load.interval`**

Hot-swap enabled: Yes

This property specifies the interval in seconds between configuration reloads. When this property is set to 0, the hot-swap mechanism is disabled.

Locale Identification Properties

```
com.sun.identity.agents.config.locale.language
com.sun.identity.agents.config.locale.country
```

- **`com.sun.identity.agents.config.locale.language`**

Hot-swap enabled: No

This property specifies the language code, such as en for English, for identifying the locale in which the site operates.

- **`com.sun.identity.agents.config.locale.country`**

Hot-swap enabled: No

This property specifies the country code for identifying the locale in which the site operates.

Organization Name Property

- **`com.sun.identity.agents.config.organization.name`**

Hot-swap enabled: No

This property specifies the organization or realm name used to authenticate the agent during runtime. The default value "`/`" identifies the root organization or realm.

Audit Log Properties

```
com.sun.identity.agents.config.audit.accesstype
com.sun.identity.agents.config.log.disposition
com.sun.identity.agents.config.remote.logfile
com.sun.identity.agents.config.local.logfile
com.sun.identity.agents.config.local.log.rotate
com.sun.identity.agents.config.local.log.size
```

- **`com.sun.identity.agents.config.audit.accesstype`**

Hot-swap enabled: No

This property specifies the access type or access types logged by the agent. The following are valid values for this property:

```
LOG_NONE
LOG_ALLOW
LOG_DENY
LOG_BOTH
```

- **`com.sun.identity.agents.config.log.disposition`**

Hot-swap enabled: Yes

This property specifies the audit log mode that the agent uses when writing audit log messages. The following are valid values for this property:

```
LOCAL
REMOTE
ALL
```

Key Properties Affecting This Property

This property is *not* used when the following property is set as shown:

```
com.sun.identity.agents.config.audit.accesstype = LOG_NONE
```

- **`com.sun.identity.agents.config.remote.logfile`**

Hot-swap enabled: Yes

This property specifies the file name used on the remote server.

Key Properties Affecting This Property

This property is *not* used when the following property is set as shown:

```
com.sun.identity.agents.config.log.disposition = LOCAL
```

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

Hot-swap enabled: Yes

This property specifies the complete path to the local audit log file to be used by the agent.

Key Properties Affecting This Property

This property is *only* used when the following property is set as shown:

```
com.sun.identity.agents.config.log.disposition = LOCAL
```

- **`com.sun.identity.agents.config.local.log.rotate`**

Hot-swap enabled: Yes

This property is a flag that indicates whether the rotation of audit log local file is enabled or disabled.

Key Properties Affecting This Property

This property is *only* used when the following property is set as shown:

```
com.sun.identity.agents.config.log.disposition = LOCAL
```

- **`com.sun.identity.agents.config.local.log.size`**

Hot-swap enabled: Yes

This property specifies the size in bytes of the local audit log file, beyond which the agent rotates the log file.

Key Properties Affecting This Property

This property is *only* used when the following property is set as shown:

```
com.sun.identity.agents.config.log.disposition = LOCAL
```

Web Service Processing Properties

```
com.sun.identity.agents.config.webservice.enable
com.sun.identity.agents.config.webservice.endpoint[]
com.sun.identity.agents.config.webservice.process.get.enable
com.sun.identity.agents.config.webservice.authenticator
com.sun.identity.agents.config.webservice.internalerror.content
com.sun.identity.agents.config.webservice.autherror.content
```

- **`com.sun.identity.agents.config.webservice.enable`**

Hot-swap enabled: Yes

This property is a flag that indicates whether web service processing is enabled or disabled.

- **`com.sun.identity.agents.config.webservice.endpoint[]`**

Hot-swap enabled: Yes

This property is a list construct for listing web application end points that represent web services.

- **`com.sun.identity.agents.config.webservice.process.get.enable`**

Hot-swap enabled: Yes

This property is a flag that indicates whether the processing of `HTTP GET` requests for web service endpoints is enabled or disabled.

- **`com.sun.identity.agents.config.webservice.authenticator`**

Hot-swap enabled: Yes

This property specifies an implementation class that can be used to authenticate web-service requests.

- **`com.sun.identity.agents.config.webservice.internalerror.content`**

Hot-swap enabled: Yes

This property specifies the name of a file that contains content used by the agent to generate an internal error fault for clients.

- **`com.sun.identity.agents.config.webservice.autherror.content`**

Hot-swap enabled: Yes

This property specifies the name of a file that contains content used by the agent to generate an authorization error fault for clients.

Access Denied URI Property

- **`com.sun.identity.agents.config.access.denied.uri`**

Hot-swap enabled: Yes

This property specifies the URI used by the agent to block unauthorized access requests. If you will not employ this property, leave it blank.

Form Login Processing Properties

```
com.sun.identity.agents.config.login.form[]
com.sun.identity.agents.config.login.error.uri[]
com.sun.identity.agents.config.login.use.internal
com.sun.identity.agents.config.login.content.file
```

- **`com.sun.identity.agents.config.login.form[]`**

Hot-swap enabled: Yes

This property is a list construct. This property is used by the agent to identify login requests and to take appropriate action. Each entry in the list should be the absolute URI of the resource specified in the `web.xml` deployment descriptor of the protected application in the element form-login-page.

- **`com.sun.identity.agents.config.login.error.uri[]`**

Hot-swap enabled: Yes

This property is a list construct. This property is used by the agent to identify error page requests and to take appropriate action. Each entry in the list should be the absolute URI of the resource specified in the `web.xml` deployment descriptor of the protected application in the element form-error-page.

- **`com.sun.identity.agents.config.login.use.internal`**

Hot-swap enabled: Yes

This property is a flag that specifies whether the agent should use internal content for handling form login requests.

- **`com.sun.identity.agents.config.login.content.file`**

Hot-swap enabled: Yes

This property specifies the name or complete path of the file used by the agent for handling form login requests.

Key Properties Affecting This Property

This property is *only* used when the following property is set as shown:

```
com.sun.identity.agents.config.login.use.internal = true
```

Local Authentication Processing Properties

```
com.sun.identity.agents.config.auth.handler[]
com.sun.identity.agents.config.logout.handler[]
com.sun.identity.agents.config.verification.handler[]
```

- **`com.sun.identity.agents.config.auth.handler[]`**

Hot-swap enabled: Yes

This property is a map construct that specifies the application specific authentication handler used by the agent to authenticate the logged on user with the deployment container for the particular application.

- **com.sun.identity.agents.config.logout.handler[]**

Hot-swap enabled: Yes

This property is a map construct that specifies the application specific logout handler used by the agent to log out the logged on user within the deployment container for the particular application.

- **com.sun.identity.agents.config.verification.handler[]**

Hot-swap enabled: Yes

This property is a map construct that specifies the application specific local verification handler used by the agent to validate the user credentials with the local repository.

Goto Parameter Name Property

- **com.sun.identity.agents.config.redirect.param**

Hot-swap enabled: Yes

This property specifies the parameter name used by the agent when redirecting the user to the appropriate authentication service. The value of this parameter is used by the authentication service to redirect the user to the original requested destination.

Login URL Property

- **com.sun.identity.agents.config.login.url[]**

Hot-swap enabled: Yes

This property is a list construct for listing the login URL (one or more) to be used by the agent to redirect incoming users without sufficient credentials to the Access Manager authentication service.

Login URL Prioritized Flag Property

- **com.sun.identity.agents.config.login.url.prioritized**

Hot-swap enabled: Yes

This property is a flag that specifies if the failover sequence for the login URL list and the CDSSO URL list is prioritized. The URL associated with the lowest index, `[0]`, has the highest priority. When set to `true`, this property turns on prioritization for both the login URL list and the CDSSO URL list, assuming each list exists. The following properties are used to create these two URL lists:

Login URL List	`com.sun.identity.agents.config.login.url[]`
CDSSO URL List	`com.sun.identity.agents.config.cdsso.cdcservlet.url[]`

For more information about enabling failover, see "Enabling Failover in J2EE Agents" on page 93.

Agent Server Properties

```
com.sun.identity.agents.config.agent.host
com.sun.identity.agents.config.agent.port
com.sun.identity.agents.config.agent.protocol
```

- **`com.sun.identity.agents.config.agent.host`**

Hot-swap enabled: Yes

This property specifies the host name that identifies the agent protected server to client browsers if the host name is different from the actual host name. If you will not employ this property, leave it blank.

- **`com.sun.identity.agents.config.agent.port`**

Hot-swap enabled: Yes

This property specifies the port number that identifies the agent protected server listening port to client browsers if the port number is different from the actual listening port. If you will not employ this property, leave it blank.

- **`com.sun.identity.agents.config.agent.protocol`**

Hot-swap enabled: Yes

The property specifies the protocol, HTTP or HTTPS , used by client browsers to communicate with the agent protected server if the protocol is different from the actual protocol used by the server.

Login Attempt Limit Property

- **`com.sun.identity.agents.config.login.attempt.limit`**

Hot-swap enabled: Yes

This property specifies the number of unsuccessful login attempts users are allowed to make during a single browser session before such attempts trigger a block on further requests. Setting the value of this property to `0` disables this feature.

URL Decode SSO Token Flag Property

- **`com.sun.identity.agents.config.sso.decode`**

Hot-swap enabled: Yes

This property is a flag that specifies whether the SSO Token needs to be URL decoded by the agent before it can be used.

SSO Cache Enable Property

- **`com.sun.identity.agents.config.amsso.cache.enable`**

Hot-swap enabled: Yes

This property is a flag that specifies whether the SSO cache is active for the agent. This cache is used through public API exposed by the agent SDK.

Cookie Reset Processing Properties

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

- **`com.sun.identity.agents.config.cookie.reset.enable`**

Hot-swap enabled: Yes

This property is a flag that specifies whether cookie reset processing is enabled or disabled.

- **`com.sun.identity.agents.config.cookie.reset.name[]`**

Hot-swap enabled: Yes

This property is a list construct for listing cookie names that are reset by the agent

Key Properties Affecting This Property

This property is *only* used when the following property is set as shown:

```
com.sun.identity.agents.config.cookie.reset.enable = true
```

- **`com.sun.identity.agents.config.cookie.reset.domain[]`**

Hot-swap enabled: Yes

This property is a map construct. The key for this map construct is a cookie name and the value for this map construct is the domain of that cookie.

Key Properties Affecting This Property

This property is used when one of the cookies listed in following property matches the key for this property:

```
com.sun.identity.agents.config.cookie.reset.name[]
```

- **`com.sun.identity.agents.config.cookie.reset.path[]`**

Hot-swap enabled: Yes

This property is a map construct. The key for this map construct is a cookie name and the value for this map construct is the path of that cookie.

Key Properties Affecting This Property

This property is used when one of the path names listed in following property matches the key for this property:

```
com.sun.identity.agents.config.cookie.reset.name[]
```

CDSSO Processing Properties

```
com.sun.identity.agents.config.cdsso.enable
com.sun.identity.agents.config.cdsso.redirect.uri
com.sun.identity.agents.config.cdsso.cdcservlet.url[]
com.sun.identity.agents.config.cdsso.clock.skew
com.sun.identity.agents.config.cdsso.trusted.id.provider[]
```

- **com.sun.identity.agents.config.cdsso.enable**

Hot-swap enabled: Yes

This property is a flag that specifies whether CDSSO processing is enabled or disabled.

- **com.sun.identity.agents.config.cdsso.redirect.uri**

Hot-swap enabled: Yes

This property specifies an intermediate URI that is used by the agent for processing CDSSO requests.

- **com.sun.identity.agents.config.cdsso.cdcservlet.url[]**

Hot-swap enabled: Yes

This property is a list construct for listing the URL of the available CDSSO controllers that can be used by the agent for CDSSO processing.

- **com.sun.identity.agents.config.cdsso.clock.skew**

Hot-swap enabled: Yes

This property specifies a time in seconds that is used by the agent to determine the validity of the CDSSO `AuthnResponse` assertion.

- **com.sun.identity.agents.config.cdsso.trusted.id.provider[]**

Hot-swap enabled: Yes

This property is a list construct for listing the Access Manager server providers, ID providers, or both to be trusted by the agent during the evaluation process.

Logout Processing Properties

```
com.sun.identity.agents.config.logout.application.handler[]
com.sun.identity.agents.config.logout.uri[]
com.sun.identity.agents.config.logout.request.param[]
com.sun.identity.agents.config.logout.introspect.enabled
com.sun.identity.agents.config.logout.entry.uri[]
```

- **com.sun.identity.agents.config.logout.application.handler[]**

Hot-swap enabled: Yes

This property is a map construct that is application specific. It identifies a handler to be used for logout processing.

- **com.sun.identity.agents.config.logout.uri[]**

Hot-swap enabled: Yes

This property is a map construct that is application specific. It identifies a request URI which indicates a logout event.

- **com.sun.identity.agents.config.logout.request.param[]**

Hot-swap enabled: Yes

This property is a map construct that is application specific. It identifies a parameter which when present in the HTTP request indicates a logout event.

- **com.sun.identity.agents.config.logout.introspect.enabled**

Hot-swap enabled: Yes

This property is a flag that allows the agent to search an HTTP request body for a logout parameter.

- **com.sun.identity.agents.config.logout.entry.uri[]**

Hot-swap enabled: Yes

This property is a map construct that is application specific. It identifies a URI to be used as an entry point after successful logout and subsequent to successful authentication if applicable.

FQDN Processing Properties

```
com.sun.identity.agents.config.fqdn.check.enable
com.sun.identity.agents.config.fqdn.default
com.sun.identity.agents.config.fqdn.mapping[]
```

- **com.sun.identity.agents.config.fqdn.check.enable**

Hot-swap enabled: Yes

This property is a flag that indicates whether FQDN checking is enabled or disabled.

- **com.sun.identity.agents.config.fqdn.default**

Hot-swap enabled: Yes

This property specifies a hostname that represents the default FQDN to be used by the agent when necessary.

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

Hot-swap enabled: Yes

This property is a map construct that specifies a mapping from the key, which is an invalid FQDN entry to its value, which is a valid FQDN entry.

Legacy User Agent Processing Properties

```
com.sun.identity.agents.config.legacy.support.enable
com.sun.identity.agents.config.legacy.user.agent[]
com.sun.identity.agents.config.legacy.redirect.uri
```

- **`com.sun.identity.agents.config.legacy.support.enable`**

Hot-swap enabled: Yes

This property is a flag that specifies whether legacy user agent support is enabled or disabled.

- **`com.sun.identity.agents.config.legacy.user.agent[]`**

Hot-swap enabled: Yes

This property is a list construct for listing user agent header values. These values identify legacy browsers. Entries in this list can contain the wild card character "*."

- **`com.sun.identity.agents.config.legacy.redirect.uri`**

Hot-swap enabled: Yes

This property specifies an intermediate URI used by the agent to redirect legacy user agent requests.

Custom Response Headers Property

- **`com.sun.identity.agents.config.response.header[]`**

Hot-swap enabled: Yes

This property is a map construct that specifies the custom headers that are set by the agent on the client browser. The key is the header name while the value represents the header value.

Redirect Attempt Limit Property

- **com.sun.identity.agents.config.redirect.attempt.limit**

Hot-swap enabled: Yes

This property specifies the number of successive single point redirects that users are allowed during a single browser session before such redirects trigger a block of the user request. Setting the value of this property to `0` disables this feature.

Port Check Processing Properties

```
com.sun.identity.agents.config.port.check.enable
com.sun.identity.agents.config.port.check.file
com.sun.identity.agents.config.port.check.setting[]
```

- **com.sun.identity.agents.config.port.check.enable**

Hot-swap enabled: Yes

This property is a flag that indicates whether port check functionality is enabled or disabled.

- **com.sun.identity.agents.config.port.check.file**

Hot-swap enabled: Yes

This property specifies the name or complete path of a file that has the content required to process requests that call for port correction.

- **com.sun.identity.agents.config.port.check.setting[]**

Hot-swap enabled: Yes

This property is a map construct of port versus protocol entries where the key is the listening port number and the value is the listening protocol used by the agent to identify requests with invalid port numbers.

Not-Enforced URI Processing Properties

```
com.sun.identity.agents.config.notenforced.uri[]
com.sun.identity.agents.config.notenforced.uri.invert
com.sun.identity.agents.config.notenforced.uri.cache.enable
com.sun.identity.agents.config.notenforced.uri.cache.size
```

- **com.sun.identity.agents.config.notenforced.uri[]**

Hot-swap enabled: Yes

This property is a list construct for listing URI for which protection is not enforced by the agent.

- **`com.sun.identity.agents.config.notenforced.uri.invert`**

Hot-swap enabled: Yes

This property is a flag that specifies whether to invert the list of URI on the not-enforced list. A value of `true` directs the agent to deny access (enforce protection) to URI on the list and to allow access (not enforce protection) to URI that are not on the list. Entries on this list can contain the wild card character "*."

Key Properties Affecting This Property

This property enforces URI on the not-enforced list, which is the list assigned to the following property:

```
com.sun.identity.agents.config.notenforced.uri[]
```

- **`com.sun.identity.agents.config.notenforced.uri.cache.enable`**

Hot-swap enabled: Yes

This property is a flag that specifies whether the caching of the not-enforced URI list evaluation results is enabled or disabled.

- **`com.sun.identity.agents.config.notenforced.uri.cache.size`**

Hot-swap enabled: Yes

This property specifies the size of the cache to be used if caching of not-enforced URI list evaluation results is enabled.

Key Properties Affecting This Property

This property is only used when the following property is set as shown:

```
com.sun.identity.agents.config.notenforced.uri.cache.enable = true
```

Not-Enforced Client IP Processing Properties

```
com.sun.identity.agents.config.notenforced.ip[]
com.sun.identity.agents.config.notenforced.ip.invert
com.sun.identity.agents.config.notenforced.ip.cache.enable
com.sun.identity.agents.config.notenforced.ip.cache.size
```

- **`com.sun.identity.agents.config.notenforced.ip[]`**

Hot-swap enabled: Yes

This property is a list construct for listing client IP addresses for which protection is not enforced by the agent.

- **`com.sun.identity.agents.config.notenforced.ip.invert`**

Hot-swap enabled: Yes

This property is a flag that specifies whether to invert the not-enforced client IP address list. A value of `true` directs the agent to deny access (enforce protection) to client IP addresses on the list and to allow access (not enforce protection) for all other client IP addresses. Entries on this list can contain the wild card character "*."

Key Properties Affecting This Property

This property enforces URI on the not-enforced IP list, which is the list assigned to the following property:

```
com.sun.identity.agents.config.notenforced.ip[]
```

- **`com.sun.identity.agents.config.notenforced.ip.cache.enable`**

Hot-swap enabled: Yes

A flag that specifies whether the caching of not-enforced IP list evaluation results is enabled or disabled.

- **`com.sun.identity.agents.config.notenforced.ip.cache.size`**

Hot-swap enabled: Yes

This property specifies the size of the cache to be used if caching of not-enforced IP list evaluation results is enabled.

Key Properties Affecting This Property

This property is only used when the following property is set as shown:

```
com.sun.identity.agents.config.notenforced.ip.cache.enable = true
```

Common Attribute Fetch Processing Properties

```
com.sun.identity.agents.config.attribute.cookie.separator
com.sun.identity.agents.config.attribute.date.format
com.sun.identity.agents.config.attribute.cookie.encode
```

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

Hot-swap enabled: Yes

This property specifies that a character be used to separate multiple values of the same attribute when it is being set as a cookie.

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

Hot-swap enabled: Yes

This property is a flag that indicates whether the value of the attribute should be URL encoded before being set as a cookie.

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

Hot-swap enabled: Yes

This property specifies the format of date attribute values used when the attribute is set as an HTTP header. This format is based on the definition provided in `java.text.SimpleDateFormat`.

Profile Attribute Processing Properties

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

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

Hot-swap enabled: Yes

This property specifies the mode used to fetch profile attributes. The following are valid values for this property:

```
NONE
HTTP_HEADER
REQUEST_ATTRIBUTE
HTTP_COOKIE
```

- **`com.sun.identity.agents.config.profile.attribute.mapping[]`**

Hot-swap enabled: Yes

This property is a map construct that specifies the profile attributes populated under specific names for the currently authenticated user. The key for this map construct is the profile attribute name and the value is the name under which that attribute is made available.

Session Attribute Processing Properties

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

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

Hot-swap enabled: Yes

This property specifies the mode used to fetch session attributes. The following are valid values for this property:

```
NONE
HTTP_HEADER
REQUEST_ATTRIBUTE
HTTP_COOKIE
```

- **`com.sun.identity.agents.config.session.attribute.mapping[]`**

Hot-swap enabled: Yes

This property is a map construct that specifies the session attributes populated under specific names for the currently authenticated user. The key for this map construct is the session attribute name and the value is the name under which that attribute is made available.

Response Attribute Processing Properties

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

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

Hot-swap enabled: Yes

This property specifies the mode used to fetch policy response attributes. The following are valid values for this property:

```
NONE
```

```
HTTP_HEADER
REQUEST_ATTRIBUTE
HTTP_COOKIE
```

- **com.sun.identity.agents.config.response.attribute.mapping[]**

Hot-swap enabled: Yes

This property is a map construct that specifies the policy response attributes to be populated under specific names for the currently authenticated user. The key for this map construct is the policy response attribute name and the value is the name under which that attribute is made available.

Bypass Principal List Property

- **com.sun.identity.agents.config.bypass.principal[]**

Hot-swap enabled: No

This property is a list construct for listing principals that are to be bypassed by the agent for authentication and search purposes.

Privileged Attribute Processing Properties

```
com.sun.identity.agents.config.default.privileged.attribute[]
com.sun.identity.agents.config.privileged.attribute.type[]
com.sun.identity.agents.config.privileged.attribute.tolowercase[]
com.sun.identity.agents.config.privileged.session.attribute[]
```

- **com.sun.identity.agents.config.default.privileged.attribute[]**

Hot-swap enabled: No

This property is a list construct for listing privileged attributes to be granted to all users who have a valid Access Manager session.

- **com.sun.identity.agents.config.privileged.attribute.type[]**

Hot-swap enabled: No

This property is a list construct for listing privileged attribute types to be fetched for each user.

- **com.sun.identity.agents.config.privileged.attribute.tolowercase[]**

Hot-swap enabled: No

This property is a map construct that specifies whether the privileged attribute types are converted to lowercase.

Key Properties Affecting This Property

This property converts the attribute types assigned to the following property to lower case:

```
com.sun.identity.agents.config.privileged.attribute.type[]
```

- **`com.sun.identity.agents.config.privileged.session.attribute[]`**

Hot-swap enabled: No

This property is a list construct for listing session property names that hold privileged attributes for the authenticated user.

Service Resolver Property

- **`com.sun.identity.agents.config.service.resolver`**

Hot-swap enabled: No

This property specifies the service resolver used by this agent.

Agent Username and Password Properties

```
com.sun.identity.agents.app.username
com.iplanet.am.service.secret
```

- **`com.sun.identity.agents.app.username`**

Hot-swap enabled: No

This property specifies the user name used by the agent to identify and authenticate itself to Access Manager before requesting any services that require such agent authentication.

- **`com.iplanet.am.service.secret`**

Hot-swap enabled: No

This property specifies the password used by the agent to identify and authenticate itself to Access Managerbefore requesting any services that require such agent authentication.

Encryption Key Properties

```
am.encryption.pwd
com.sun.identity.client.encryptionKey
```

- **`am.encryption.pwd`**

Hot-swap enabled: No

This property specifies a global encryption key used when applications use client SDK API. This encryption key is used to secure data globally by all Access Manager server instances and by clients.

- **`com.sun.identity.client.encryptionKey`**

Hot-swap enabled: No

This property specifies the encryption key used to encrypt the agent profile password as it is stored in the J2EE agent. The agent profile password is encrypted in a different manner in Access Manager. This encryption key is not shared with Access Manager or with other clients.

Debug Service Properties

```
com.iplanet.services.debug.level
com.iplanet.services.debug.directory
```

- **`com.iplanet.services.debug.level`**

Hot-swap enabled: No

This property specifies the debug level to be used. The following are valid values for this property:

```
off
error
warning
message
```

- **`com.iplanet.services.debug.directory`**

Hot-swap enabled: No

This property specifies the complete path to the directory where debug files are to be stored by the agent.

SSO Token Cookie Name Property

- **`com.iplanet.am.cookie.name`**

Hot-swap enabled: No

This property specifies the name of the SSO token cookie used betweenAccess Manager and the agent.

Naming Service URL Property

- **`com.iplanet.am.naming.url`**

Hot-swap enabled: No

This property specifies the naming service URL (one or more) that can be used by the system for naming lookups. Multiple URL can be specified for this property as a string. URL are separated from one another in the string by a single space character.

Session Client Properties

```
com.iplanet.am.notification.url
com.iplanet.am.session.client.polling.enable
com.iplanet.am.session.client.polling.period
```

- **`com.iplanet.am.notification.url`**

Hot-swap enabled: No

This property specifies the notification URL to be used by the agent to receive session notifications.

- **`com.iplanet.am.session.client.polling.enable`**

Hot-swap enabled: No

This property is a flag that specifies whether the session client uses polling for updating session information instead of depending upon server notifications.

- **`com.iplanet.am.session.client.polling.period`**

Hot-swap enabled: No

This property specifies the time in seconds after which the session client requests an update of cached session information from the server.

Encryption Provider Property

- **`com.iplanet.security.encryptor`**

Hot-swap enabled: No

This property specifies the encryption provider implementation to be used by the agent.

User Data Cache Update Time Property

- **`com.iplanet.am.sdk.remote.pollingTime`**

Hot-swap enabled: No

This property specifies the cache update time in minutes for user management data if a notification URL is not provided.

Key Properties Affecting This Property

This property is used if a notification URL is *not* specified with the following property:

```
com.iplanet.am.notification.url
```

Service Data Cache Update Time Property

- **`com.sun.identity.sm.cacheTime`**

Hot-swap enabled: No

This property specifies the cache update time in minutes for service configuration data if a notification URL is not provided.

Key Properties Affecting This Property

This property is used if a notification URL is *not* specified with the following property:

```
com.iplanet.am.notification.url
```

SAML Service Properties

```
com.iplanet.am.localserver.protocol
com.iplanet.am.localserver.host
com.iplanet.am.localserver.port
```

- **`com.iplanet.am.localserver.protocol`**

Hot-swap enabled: No

This property specifies the server protocol to be used for SAML service.

- **`com.iplanet.am.localserver.host`**

Hot-swap enabled: No

This property specifies the server host to be used for SAML service.

- **`com.iplanet.am.localserver.port`**

Hot-swap enabled: No

This property specifies the server port to be used for SAML service.

Authentication Service Properties

```
com.iplanet.am.server.protocol
com.iplanet.am.server.host
com.iplanet.am.server.port
```

- **`com.iplanet.am.server.protocol`**

Hot-swap enabled: No

This property specifies the protocol to be used by Authentication Service.

- **`com.iplanet.am.server.host`**

Hot-swap enabled: No

This property specifies the host to be used by Authentication Service.

- **`com.iplanet.am.server.port`**

Hot-swap enabled: No

This property specifies the port to be used by Authentication Service.

Policy Client Properties

```
com.sun.identity.agents.server.log.file.name
com.sun.identity.agents.logging.level
com.sun.identity.agents.notification.enabled
com.sun.identity.agents.notification.url
com.sun.identity.agents.polling.interval
com.sun.identity.policy.client.cacheMode
com.sun.identity.policy.client.booleanActionValues
com.sun.identity.policy.client.resourceComparators
com.sun.identity.policy.client.clockSkew
```

- **`com.sun.identity.agents.server.log.file.name`**

Hot-swap enabled: No

This property specifies the name of the log file for logging messages to Access Manager.

- **`com.sun.identity.agents.logging.level`**

Hot-swap enabled: No

This property specifies the level of remote policy logging. The following are valid values for this property:

```
ALLOW
DENY
BOTH
NONE
```

- **`com.sun.identity.agents.notification.enabled`**

Hot-swap enabled: No

This property is a flag that specifies whether notifications are enabled or disabled for the remote policy client.

- **`com.sun.identity.agents.notification.url`**

Hot-swap enabled: No

This property specifies the notification URL for the remote policy client.

Key Properties Affecting This Property

This property is used if notification is enabled for a remote policy client property, which occurs when the following property is set as shown:

```
com.sun.identity.agents.notification.enabled = true
```

- **`com.sun.identity.agents.polling.interval`**

Hot-swap enabled: No

This property specifies the duration in minutes after which the cached entries are refreshed by the remote policy client.

- **`com.sun.identity.policy.client.cacheMode`**

Hot-swap enabled: No

This property specifies the mode of caching to be used by the remote policy client. The following are valid values for this property:

```
subtree
self
```

The `subtree` value is preferable for a small number of policy rules. In all other cases, the `self` value is preferable.

- **`com.sun.identity.policy.client.booleanActionValues`**

Hot-swap enabled: No

This property specifies boolean action values for policy action names. Assign values to this property using the following format:

```
serviceName|actionName|trueValue|falseValue
```

- **`com.sun.identity.policy.client.resourceComparators`**

Hot-swap enabled: No

This property specifies resource comparators to be used for different service names.

- **`com.sun.identity.policy.client.clockSkew`**

Hot-swap enabled: No

This property specifies the time in seconds which is allowed to accommodate the time difference between the Access Manager machine and the remote policy client machine.

APPENDIX C

Troubleshooting a J2EE Agent Deployment in Policy Agent 2.2

This appendix explains how you can resolve problems that you might encounter while deploying or using J2EE agents.

Be sure to also check the *Sun Java System Access Manager Policy Agent 2.2 Release Notes*, to see if the problem that you encounter is a known limitation of the agent. If workarounds are available for such problems, they are provided in the release notes.

J2EE Agent Troubleshooting Instructions

This section includes various symptoms. Each symptom is accompanied by one or more possible causes. Each possible cause is accompanied by a troubleshooting solution.

1. Symptom: The agent does not require users to login before access is granted to the application.

Possible Cause	Troubleshooting Solution
1–1) The application has not been configured to use the agent.	1–1) For information about deploying the agent application, see Chapter 4, "Post-Installation Tasks of Policy Agent 2.2 for Apache Tomcat 6.0."

1–2) The application fails to create an SSO token because a valid agent profile does not exist. J2EE agents authenticate with Access Manager using an agent profile, which is created in Access Manager Console.	1–2) Using Access Manager Console, ensure that a valid agent profile exists. Using the command line, encrypt the agent profile password with the `agentadmin --encrypt` command. In the J2EE `AMAgent.properties` configuration file, ensure that the following properties have the updated agent profile name and password:
	`com.sun.identity.agents.app.username =`
	`com.iplanet.am.service.secret=`

1–3) The resources match entries in the not-enforced list.	1–3) Make sure that the resources being accessed do not match the entries in the not-enforced list, and ensure that the list is not empty or inverted.

1. Symptom: The agent does not require users to login before access is granted to the application.

Possible Cause	Troubleshooting Solution
1–4) The agent filter mode is set to `NONE`.	1–4) Change the agent filter mode to `ALL` or `J2EE_POLICY` as necessary.

2. Symptom: The agent denies access to all requests.

Possible Cause	Troubleshooting Solution
2–1) The agent and Access Manager have been installed on the same machine and the browser might not be setting the HOST header correctly when redirected from Access Manager to the agent.	2–1) Enable port check functionality. For information about enabling port check functionality, see "Enabling Port Check Functionality in J2EE Agents" on page 104.
2–2) The deployment container is running as a user who does not have write privileges to the audit log directory of the agent.	2–2) Refer to the path specified in the J2EE agent `AMAgent.properties` configuration file for the agent's local audit file and grant the necessary write permissions for the user of the deployment container process.
2–3) The agent filter is configured for a mode that enforces URL policies and no applicable URL policies have yet been defined in Access Manager.	2–3) Define the appropriate URL policies in Access Manager.
2–4) The agent filter is configured for a mode that enforces URL polices and the system time on the agent machine is not in sync with the system time on the Access Manager machine.	2–4) Synchronize the time on the agent machine with the time on the Access Manager machine.
2–5) The agent filter is configured for a mode that does not support J2EE polices and the resources being accessed are protected by declarative security constraints.	2–5) Change the agent filter mode to a mode that supports J2EE policy such as ALL or J2EE_POLICY.
2–6) The agent filter is configured for a mode that supports J2EE polices but they are being negatively evaluated by the agent.	2–6) Change the agent filter mode to a mode that supports J2EE policy such as ALL or J2EE_POLICY.
2–7) The agent is unable to validate user's session token issued by Access Manager.	2–7) Ensure that the agent is installed on the same domain that is specified as the cookie domain in Access Manager. If not, enable CDSSO functionality. If that is not the case, try changing the value of the following property: `com.sun.identity.agents.config.sso.decode`

2. Symptom: The agent denies access to all requests.

Possible Cause	Troubleshooting Solution
2–8) The agent is configured for CDSSO and the validity time of the authorization response is smaller than the processing time required by the agent.	2–8) Set an appropriate value for the following property: `com.sun.identity.agents.config.cdsso.clock.skew`
2–9) The Login URL specified in the J2EE agent `AMAgent.properties` configuration file is not reachable by the agent.	2–9) Ensure that the Access Manager Login URL is reachable from the machine where the agent is installed.
2–10) The Access Manager is installed with SSL and the agent cannot communicate with it correctly.	2–10) Install the appropriate root CA certificate in the keystore used by the deployment container on which the agent is installed.

3) Symptom: The agent fails to evaluate J2EE declarative security policies or J2EE programmatic security API for the protected applications.

Possible Cause	Troubleshooting Solution
3–1) The protected application does not have the agent filter installed.	3–1) Redeploy the application with the agent filter installed and the log-in configuration added. For more information, see Chapter 4, "Post-Installation Tasks of Policy Agent 2.2 for Apache Tomcat 6.0."
3–2) The agent filter is operating in a mode that does not support J2EE policies.	3–2) Change the agent filter mode to either ALL or J2EE_POLICY.

3–3) An invalid password was specified for the agent profile user during the agent installation. J2EE agents authenticate with Access Manager using an agent profile, which is created in Access Manager Console.	3–3) Using Access Manager Console, ensure that a valid agent profile exists. Using the command line, encrypt the agent profile password with the `agentadmin --encrypt` command. In the J2EE `AMAgent.properties` configuration file, ensure that the following properties have the updated agent profile name and password:
	`com.sun.identity.agents.app.username =`
	`com.iplanet.am.service.secret=`

3–4) The specified role-to-principal mapping is incorrect.	3–4) Ensure that the specified role-to-principal mapping for the protected application is correct and maps to actual users, actual roles, or both as they exist in Access Manager.

4) Symptom: Accessing a protected resource results in an HTTP 404 not found error.

Possible Cause	Troubleshooting Solution
4–1) The agent and Access Manager have been installed on the same machine and the browser being used might not be setting the HOST header correctly when redirected from Access Manager to the agent.	4–1) Enable Port Check Functionality. For more information about performing this task, see "Enabling Port Check Functionality in J2EE Agents" on page 104.

4) Symptom: Accessing a protected resource results in an HTTP 404 not found error.

Possible Cause	Troubleshooting Solution
4–2) The resource is protected by a J2EE declarative security constraint which is not being evaluated correctly and the server is trying to display `form-error-page`, which does not exist within the application.	4–2) Make sure that the resource specified by `form-error-page` in the web application's `web.xml` deployment descriptor actually exists within the application.
4–3) The resource is being accessed by a legacy browser such as Netscape 4.x and during the installation of the agent no valid value was entered for agent application URI field.	4–3) Reinstall the agent and specify a valid agent application URI value.
4–4) The agent is operating in the CDSSO mode and no valid value was entered for the primary application context path field during agent installation.	4–4) Reinstall the agent and specify a valid primary application context path.

5) Symptom: When Access Manager is SSL Enabled, the agent denies access to all resources.

Possible Cause	Troubleshooting Solution
5–1) The agent does not have the root CA certificate of the signer of the certificate used by Access Manager.	5–1) Install the appropriate root CA certificate in the keystore used by the deployment container on which the agent is installed.

6) Symptom: An access denied message is issued when the agent is in J2EE_Policy mode.

Possible Cause	Troubleshooting Solution
6–1) The requested resource that expects to use the attributes is in the not-enforced list of the agent.	6–1) Change the not-enforced list so that the requested resource is enforced. The agent does not provide support for LDAP attributes for resources that are not-enforced.
6–2) The agent is unable to fetch user roles from Access Manager.	6–2) Ensure that the following property is set correctly in the J2EE agent `AMAgent.properties` configuration file: `com.sun.identity.agents.config.organization.name`

7) Symptom: The Access Manager logs indicate that Access Manager is unable to send notifications to the deployment container protected by the agent.

Possible Cause	Troubleshooting Solution
7–1) The agent is installed on a server with the preferred listening protocol set to HTTPS and the root CA certificate for the signer of the agent's server certificate is not available in the keystore used by Access Manager.	7–1) Add the root CA certificate for the signer of agent's server certificate to the keystore used by Access Manager.

7) Symptom: The Access Manager logs indicate that Access Manager is unable to send notifications to the deployment container protected by the agent.

Possible Cause	Troubleshooting Solution
7–2) No valid value was entered for agent application URI during agent installation.	7–2) Reinstall the agent and specify a valid agent application URI. For example, `/agentapp`. The same URI should be used as a context root for deploying the agent application through the administration console of the container.

8) Symptom: The following warning messages appear on the console window from which the deployment container is started or in the deployment container logs:

```
Bad level value for property: com.iplanet.services.debug.level
Bad level value for property: com.sun.identity.agents.logging.level
```

Possible Cause	Troubleshooting Solution
8–1) These messages are logged by the JDK 1.4 logging framework in use, which treats the J2EE agent `AMAgent.properties` configuration file as the logging configuration file.	8–1) These messages are harmless and can be safely ignored.

Index