Sun Java System Access Manager 7.1 Developer's Guide

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

Part No: 819–4675–10
March 2007

080701@20490

Contents

Preface

Sun Java™ System Access Manager is a component of the Sun Java Enterprise System (Java ES), a set of software components that provide services needed to support enterprise applications distributed across a network or Internet environment. The *Sun Java System Access Manager 7.1 Developer's Guide* provides information about using the Access Manager Java application programming interfaces (APIs) and service preprogramming interfaces (SPIs).

For information about using the Access Manager C-APIs, see Chapter 1, "The C Application Programming Interface Files," in *Sun Java System Access Manager 7.1 C API Reference* in the document *Sun Java System Access Manager 7.1 C API Reference*.

Before You Read This Book

This book is intended for use by IT administrators and software developers who implement a web access platform using Sun Java System servers and software. Readers of this guide should be familiar with the following concepts and technologies:

- Deployment platform: Solaris™ or Linux operating system
- Web container that will run Access Manager: Sun Java System Application Server, Sun Java System Web Server, BEA WebLogic, or IBM WebSphere Application Server
- Technical concepts: Lightweight Directory Access Protocol (LDAP), Java technology, JavaServer Pages™ (JSP) technology, HyperText Transfer Protocol (HTTP), HyperText Markup Language (HTML), and eXtensible Markup Language (XML)

Related Books

Related documentation is available as follows:

- "Access Manager Core Documentation" on page 9
- "Sun Java Enterprise System Product Documentation" on page 11

Access Manager Core Documentation

The Access Manager core documentation set contains the following titles:

- The *Sun Java System Access Manager 7.1 Release Notes* will be available online after the product is released. It gathers an assortment of 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.
- The *Sun Java System Access Manager 7.1 Technical Overview* provides an overview of how Access Manager components work together to consolidate access control functions, and to protect enterprise assets and web-based applications. It also explains basic Access Manager concepts and terminology.
- The *Sun Java System Access Manager 7.1 Deployment Planning Guide* provides planning and deployment solutions for Sun Java™ System Access Manager based on the solution life cycle.
- The *Sun Java System Access Manager 7.1 Postinstallation Guide*provides information about basic Access Manager configuration tasks you must perform immediately after running the Java Enterprise System installer.
- The *Sun Java System Access Manager 7.1 Performance Tuning and Troubleshooting Guide* provides information on how to tune Access Manager and its related components for optimal performance.
- The *Sun Java System Access Manager 7.1 Administration Guide* describes how to use the Access Manager console as well as manage user and service data via the command line interface.
- The *Sun Java System Access Manager 7.1 Federation and SAML Administration Guide*provides information about the Federation module based on the Liberty Alliance Project 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.
- The *Sun Java System Access Manager 7.1 Developer's Guide* (this guide) offers information on how to customize Access Manager and integrate its functionality into an organization's current technical infrastructure. It also contains details about the programmatic aspects of the product and its API.
- The *Sun Java System Access Manager 7.1 C API Reference* provides summaries of data types, structures, and functions that make up the public Access Manager C APIs.
- The *Sun Java System Access Manager 7.1 Java API Reference* (part number 819-2141) provides information about the implementation of Java packages in Access Manager.
- The *Sun Java System Access Manager Policy Agent 2.2 User's Guide* provides an overview of the policy functionality and the policy agents available for Access Manager.

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 Enterprise System documentation web site. Updated documents will be marked with a revision date.

Sun Java Enterprise System Product Documentation

Useful information can be found in the documentation for the following products:

- Directory Server
- Web Server
- Application Server
- Web Proxy Server

Related Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

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

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (`http://www.sun.com/documentation/`)
- Support (`http://www.sun.com/support/`)
- Training (`http://www.sun.com/training/`)

Typographic Conventions

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

TABLE P–1 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`

TABLE P–1 Typographic Conventions *(Continued)*

Typeface	Meaning	Example
AaBbCc123	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file. **Note:** Some emphasized items appear bold online.

Shell Prompts in Command Examples

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

TABLE P–2 Shell Prompts

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

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 book or at the top of the document.

For example, the title of this book is *Sun Java System Access Manager 7.1 Developer's Guide*, and the part number is 819-4675–10.

◆ ◆ ◆ CHAPTER 1

Using the Client SDK

The Sun Java™ System Access Manager 7.1 Client SDK package provides Access Management Java libraries for implementing stand-alone applications and web applications. You can use the Client SDK interfaces in your applications to take advantage of Access Manger services such as authentication, Single Sign-On (SSO), authorization, auditing and logging, user management, and Security Assertion Markup Language (SAML). The client SDK libraries communicate with Access Manager using XML (SOAP) over HTTP or HTTPS.

This chapter contains the following topics:

How the Client SDK Works

The Access Manager Client SDK is a streamlined version of the Access Manager SDK, and includes only the client-side classes and configuration properties you need to connect to Access Manager services. The Client SDK is contained in a smaller jar file, and eliminate the dependency on connections to Directory Server when developing and deploying client applications. The Client SDK and client applications communicate with the Access Manager server. Only the Access Manager server communicates directly with the Directory Server.

When you install the Access Manager server, the Client SDK is contained in the following file:

AccessManager-base`/SUNWam/lib/amclientsdk.jar`

The following table summarizes items included in the Client SDK.

TABLE 1–1 Contents of *AccessManager-base*/SUNWam/amclientsdk.jar

File	Description
README.clientsdk	ASCII version of this chapter. Contains information on installing and using Access Manager client SDK.
lib/amclientsdk.jar	Client SDK for stand-alone applications.
amclient.war	Archive of Access Manager samples, web applications, and Javadoc.
Makefile.clientsdk	Defines objects and parameters for building sample properties, stand-alone samples and web applications.

JDK and CLASSPATH Requirements

The Client SDK requires J2SE 1.4.2 .

Note – Both amclientsdk.jar and servlet.jar are required in the CLASSPATH.

Installing the Client SDK

Before you can install the Access Manager Client SDK, the Access Manager server must be running on a remote server. You will also need to have ready the following information about the remote installation:

- Protocol used by the web contain instance on which the Access Manager server is deployed.
- Fully qualified domain name of the host on which Access Manager server is deployed.
- Port on which the Access Manager server is running.
- Deployment URI for the services web application (by default amserver).
- Password encryption key used by the Access Manager server.

Two methods exist for installing the Client SDK. You can automatically install the Client SDK using the Java Enterprise System 5 installer, or you can manually deploy the Client SDK WAR file.

Using the Java Enterprise System 5 Installer

If you install the Access Manager Client SDK by running the Java Enterprise System 5 installer, you must choose the Access Manager Client SDK install option. For detailed information, see Chapter 10, “Deploying the Client SDK,” in *Sun Java System Access Manager 7.1 Postinstallation Guide*.

▼ To Install the Client SDK on the Windows Platform

On the Windows platform, you must use the "Configure Manually After Installation" option in the Java Enterprise System 5 installer. For detailed information, see "To Install and Configure the Access Manager Client SDK" in *Sun Java System Access Manager 7.1 Postinstallation Guide*. The following is an overview of the steps you must follow

1. **Invoke the Java Enterprise System 5 installer.**
   a. **When prompted for Setup Type, choose Custom.**
   b. **When prompted for Product Configuration, choose "Configure manually after install."**
   c. **When prompted for Products to Install, select the web container (Web Server or Application Server) and Access Manager Client SDK.**
2. **Configure and start the Web container.**
3. **Edit the file** `AccessManager-base\identity\setup\AMConfigurator.properties`**.**
   See "To Install and Configure the Access Manager Client SDK" in *Sun Java System Access Manager 7.1 Postinstallation Guide*for detailed information.
4. **Run the** `amconfig.bat` **command:**
   `# AccessManager-base\identity\setup\amconfig.bat`

▼ To Install the Client SDK on Sun Java System Web Server and Sun Java System Application Server

The following is an overview of the steps you must follow. For detailed information, see Chapter 10, "Deploying the Client SDK," in *Sun Java System Access Manager 7.1 Postinstallation Guide*.

1. **Invoke the JES installer.**
2. **Select the web container (Web Server or Application Server) and Access Manager client SDK.**
3. **Choose "Configure Now."**
4. **Answer the questions provided by the installer.**
5. **After the installation is complete, restart the web container.**

▼ To Install the Client SDK on BEA WebLogic Server and IBM WebSphere Server

The third party web container should already be installed and started.

The following is an overview of the steps you must follow. For detailed information, see Chapter 10, "Deploying the Client SDK," in *Sun Java System Access Manager 7.1 Postinstallation Guide*.

1. **Invoke the JES installer.**
2. **Select Access Manager client SDK.**
3. **Choose "Configure Later."**
4. **After the installation, edit the** `amsamplesilent` **file.**
   a. **Change the** `DEPLOY_LEVEL` **value to 9.**
   b. **Supply values for all the commented settings.**
   c. **Change the value for** `BASEDIR` **if needed.**
   d. **Edit the web container value (for example,** `WL8_*` **or** `WAS51*`**) to perform the deployment.**
5. **Run** `amconfig` **with the edited** `amsamplesilent` **file.**
6. **After the configuration is complete, restart the web container.**

▼ To Manually Install the Client SDK

Before You Begin

- The Access Manager server which will be used by the client SDK must be up and running, and you must know the URL for accessing this server.
- The machine where the client SDK will be installed must have an Access Manager supported web container installed. Examples of Access Manager supported web containers are Sun Java System Web Server 6.1 sp5, Sun Java System Application Server 8.1, BEA WebLogic Server 8.1 sp4, and IBM Websphere Application Server 5.1.1.5.
- The web container instance on which the client SDK will be deployed must be up and running.
- The client SDK machine must have access to the Access Manager client SDK package `SUNWamclnt` through the Java Enterprise System 5 bits or through some other means.

1. **Create a package administration file.**

   Using a text editor, add the following contents to this file.

   ```
   mail=
   instance=unique
   partial=nocheck
   runlevel=nocheck
   idepend=nocheck
   rdepend=nocheck
   space=nocheck
   setuid=nocheck
   conflict=nocheck
   action=nocheck
   basedir=ClientSDK-base-directory
   ```

   In this example, the package administration file is named `/usr/tmp/pkgadmin`.

   The value for `basedir`is the directory in which you want to install the Access Manager client SDK.

2. **Create a package response file named `/usr/tmp/pkgresp`.**

   Using a text editor, place the following three lines (a single `y` on each line) in this file.

   ```
   y
   y
   y
   ```

3. **In the Access Manager package directory, use the `pkgadd` utility to install the `SUNWamclnt` package:**

   `cd` *JES5-Image-root*/*OperatingSystem-Architecture*`/Product/identity_svr/Packages`

   ```
   pkgadd -n -a /usr/tmp/pkgadmin -d . -r /usr/tmp/pkgresp -R / SUNWamclnt
   ```

4. **In the directory in which you installed the Access Manager client SDK package, make a copy of the file `Makefile.clientsdk`.**

   The directory in which you installed the Access Manager client SDK package should be the same as the value you used for *basedir* in the package administration file in step 1a.

   On Solaris: `cd` *ClientSDK-base-directory*`/SUNWam`

   ```
   cp Makefile.clientsdk Makefile.clientsdk.orig
   ```

   On Linux: `cd` *ClientSDK-base-directory*`/identity`

   ```
   cp Makefile.clientsdk Makefile.clientsdk.orig
   ```

▼ To Configure the Client SDK

1 In `Makefile.clientsdk`, edit the following parameters:

JAVA_HOME	Use the following path: `/usr/jdk/entsys-j2se`
SERVER_HOSTNAME	The fully-qualified domain name of the Access Manager server.
SERVER_PROTOCOL	If the Access Manager server is SSL-enabled, change this value to HTTPS.
SERVER_PORT	The port number on which the Access Manager server is running.
ENCRYPTION_KEY	This value must be the same value used for the Access Manager server. You can obtain the value by running one of the following commands on the Access Manager server: On Solaris `grep pwd /etc/opt/SUNWam/config/AMConfig.properties` On Linux `grep pwd /etc/opt/sun/identity/config/AMConfig.properties`
DEBUG_DIR	(Optional) If you don't want the debug logs stored in the tmp directory, then change this value to the directory where you want debug logs to be created.

2 Run the make or gmake command:

```
make -f Makefile.clientsdk
```

This step generates a sample properties file in the directory `temp`, standalone samples in the directory `clientsdk-samples` and a deployable war file, `amclientwebapps.war`. The following table summarizes the items included in the WAR file.

File	Description
`index.html`	Instructions for installing and using the Client SDK packages
`WEB-INF/web.xml`	Client SDK for stand-alone applications
`WEB-INF/classes/AMClient.properties`	Archive of Access Manager samples, web applications, and Javadoc
`WEB-INF/classes/*.classes`	File for building stand-alone samples and web applications
`WEB-INF/docs`	Javadoc (Public Client SDK APIs)
`WEB-INF/samples`	Sample stand-alone programs
`WEB-INF/webapps`	Sample web applications

▼ To Deploy amclientwebapps.war

1 **Create a deployment directory for** `amclientwebapps.war`**.**

On Solaris `mkdir -p` *ClientSDK-base-directory*`/SUNWam/web-src/clientsdk`

On Linux `mkdir -p` *ClientSDK-base-directory*`/identity/web-src/clientsdK`

2 **On the web container instance where you want to use the Access Manager client SDK, deploy the** `amclientwebapps.war` **file. See the following examples:**

Sun Java Enterprise Web Server on Solaris or Linux

Use the `wdeploy` command to deploy `amclientwebapps.war` with a URI of `/amcilentwebapps` on the Web Server instance `https-`*clientSDKinstance*. Example:

WebServer-base-directory`/bin/https/httpadmin/bin/wdeploy deploy -u /amclientwebapps -i` *https-clientSDKinstance* `-v` *https-clientsdkinstance* `-d` *ClientSDK-base-directory*`/SUNWam/web-src/clientsdk` *clientSDK-base-directory*`/SUNWam/amclientwebapps.war`

Sun Java Enterprise Application Server on Solaris

Using the `asadmin` command to deploy `amclientwebapps.war` with a URI of `/amclientwebapps` on the application server instance *clientsSDKinstance*. Example:

ApplicationServer-base-directory`/appserver/bin/asadmin deploy -user` *Admin-User-ID* `--host` *ApplicationServer-instanceHost* `--port` *ApplicationServer-Admin-Port* `--contextroot amclientwebapps -name amcilentwebapps -target` *clientSDKinstance* *ClientSDK-base-directory*`/SUNWam/amclientwebapps.war`

Be sure to use the fully qualified host name for *ApplicationServer-instanceHost*.

Enter the Application Server administration password when prompted.

Sun Java Enterprise Application Server on Linux

Using the `asadmin` command to deploy `amclientwebapps.war` with a URI of `/amclientwebapps` on the application server instance *clientsSDKinstance*. Example:

ApplicationServer-base-directory`/bin/asadmin deploy -user` *Admin-User-ID* `--host` *ApplicationServer-instanceHost* `--port` *ApplicationServer-Admin-Port* `--contextroot amclientwebapps -name amcilentwebapps -target` *clientSDKinstance* *ClientSDK-base-directory*`/SUNWam/amclientwebapps.war`

Be sure to use the fully qualified host name for *ApplicationServer-instanceHost*.

Enter the Application Server administration password when prompted.

If you are deploying the client SDK on a third-party web container such as BEA WebLogic Server or IBM WebSphere Application Server, then see the documentation that comes with that product.

3 **Restart the web container instance on which the Access Manager client SDK was deployed.**

If the full server instance being accessed by the client SDK is SSL-enabled, then you must install the root CA certificate of the server's certificate in the web container's JVM-wide `cacerts` keystore. Alternatively, you can create a keystore on the client SDK machine to contain the server's root CA certificate. Then add the necessary JVM options to the client SDK's web container instance to locate the newly created keystore.

Initializing the Client SDK

Before Access Manager Client SDK can communicate with Access Manager Server, you must initialize some properties in the client SDK. You can set these properties in one of three ways:

- "Using a Properties File" on page 27
- "Using the Java API" on page 27
- "Setting Individual Properties" on page 28

Properties Used by the ClientSDK

Access Manager properties are contained in the `AMConfig.properties` file. Generate the `AMConfig.properties` for the Client SDK by running the following command:

```
# make -f Makefile.clientsdk properties
```

The following sections describe the properties expected by the Access Manager Client SDK. A client application deployed within a servlet container can register for changes to session, user attributes and policy decisions. These properties must be set to receive such notifications.

Naming URL Properties

`com.iplanet.am.naming.url`
This is a required property. The value of this property represents the URL where the Client SDK would retrieve the URLs of Access Manager internal services. This is the URI for the Naming Service. Example:

`com.iplanet.am.naming.url=http://`*AcceessManager-HostName.domain_name:port*`/amserver/namin`

`com.iplanet.am.naming.failover.url`
This property can be used by any remote SDK application that wants failover in, for example, session validation or getting the service URLs. Example:

`com.iplanet.am.naming.failover.url=http://`*AcceessManager-HostName.domain_name*:*port*`/amserver/failover`

Debug Properties

`com.iplanet.services.debug.level`
Specifies the debug level. Possible values are levels are: `off`, `error` , `warning`, or `message`.

`com.iplanet.services.debug.directory`
The value of this property is the output directory for the debug information. This directory should be writable by the server process. Example:

```
com.iplanet.services.debug.directory=/var/opt/SUNWam/debug
```

Notification URL Properties

`com.iplanet.am.notification.url`
The value of this property is the URI of the Notification Service running on the host machine where you installed the Client SDK. Example:

```
com.iplanet.am.notification.url=
http://clientSDK_host.domain_name:port/amserver/notificationservice
```

`com.sun.identity.agents.notification.enabled`
Enables or disables notifications for remote policy API. Example:

```
com.sun.identity.agents.notification.enabled=false
```

`com.sun.identity.agents.notification.url`
Defines the notification URL for remote policy API.

Security Credentials Properties

`com.sun.identity.agents.app.username`
Reads configuration data. Example:

```
com.sun.identity.agents.app.username=APPLICATION_USER
```

`com.iplanet.am.service.password`
Reads configuration data. Example:

```
com.iplanet.am.service.password=APPLICATION_PASSWD
```

`com.iplanet.am.service.secret`
Contains the encrypted value of the password. . Example:

```
com.iplanet.am.service.secret=ENCRYPTION_KEY
```

Encryption Properties

`am.encryption.pwd`
This key is needed to decrypt passwords stored in the SMS configuration. Example:

```
am.encryption.pwd=ENCRYPTION_KEY
```

`com.sun.identity.client.encryptionKey`
Encryption key that will be used to encrypt and decrypt data used locally within the client. Example:

```
com.sun.identity.client.encryptionKey=ENCRYPTION_KEY_LOCAL
```

`com.iplanet.security.encryptor`
Property to set JCE as the default encryption classes.

```
com.iplanet.security.encryptor=com.iplanet.services.util.JCEEncryption
```

Cache Update Properties

`com.sun.identity.sm.cacheTime`
Cache update time (in minutes) for service configuration data if notification URL is not provided. Example:

```
com.sun.identity.sm.cacheTime=1
```

`com.iplanet.am.sdk.remote.pollingTime`
Cache update time (in minutes) for user management cache if notification URL is not provided. Example: `com.iplanet.am.sdk.remote.pollingTime=1`

Authentication Service Properties

`com.iplanet.am.server.protocol`
Server protocol to be used by Authentication Service. Example:

```
com.iplanet.am.server.protocol=SERVER_PROTOCOL
```

`com.iplanet.am.server.host`
Server host to be used by Authentication Service. Example:

```
com.iplanet.am.server.host=SERVER_HOST
```

`com.iplanet.am.server.port`
Server port to be used by Authentication Service. Example:

```
com.iplanet.am.server.port=SERVER_PORT
```

Cookie Property

`com.iplanet.am.cookie.name`
Indicates the Access Manager cookie name. Example:

```
com.iplanet.am.cookie.name=AM_COOKIE_NAME
```

Session Service Properties

`com.iplanet.am.session.client.polling.enable`
Example:

```
com.iplanet.am.session.client.polling.enable=true
```

`com.iplanet.am.session.client.polling.period`
Example:

```
com.iplanet.am.session.client.polling.period=180
```

Certificate Database Properties

`com.iplanet.am.admin.cli.certdb.dir`
Identifies the certificate database directory path for initializing the JSS Socket Factory when the Access Manager web container is configured for SSL. Example:

```
com.iplanet.am.admin.cli.certdb.dir=
CONTAINER_CERTDB_DIR
```

`com.iplanet.am.admin.cli.certdb.passfile`
Identifies the certificate database password file for initializing the JSS Socket Factory when the Access Manager web container is configured for SSL. Example:

```
com.iplanet.am.admin.cli.certdb.passfile=
BASEDIR/PRODUCT_DIR/config/.wtpass
```

`com.iplanet.am.admin.cli.certdb.prefix`
Identifies the certificate database prefix for initializing the JSS Socket Factory when the Access Manager web container is configured for SSL. Example:

```
com.iplanet.am.admin.cli.certdb.prefix=
CONTAINER_CERTDB_PREFIX
```

Policy Client Properties

`com.sun.identity.agents.server.log.file.name`
Specifies file name for the policy decision log file. Example:

```
com.sun.identity.agents.server.log.file.name=amRemotePolicyLog
```

`com.sun.identity.agents.logging.level`
Possible values for policy decision logging level are `NONE`, `ALLOW`, `DENY`, `BOTH`, and `DECISION`.

`com.sun.identity.agents.notification.enabled`
Enables Notification URL for updating cache. Default value is `false`.

`com.sun.identity.agents.notification.url`
Specifies use of Notification URL for updating cache.
Example:`com.sun.identity.agents.notification.url=NOTIFICATION_URL`

`com.sun.identity.agents.polling.interval`
Cache time in minutes. Example:

```
com.sun.identity.agents.polling.interval=3
```

`com.sun.identity.policy.client.cacheMode`
Information to cache. Possible value are `subtree` or `self`.

`com.sun.identity.policy.client.usePre22BooleanValues`
Define and set this property to `false` if you do not want to use Boolean values. The default value is `true` if the property is not defined.

Monitoring Framework Property

`com.sun.identity.monitoring=off` Explicitly disables Java Enterprise System (JES) monitoring services in the sample client applications.

Remote Client SDK Property

`com.iplanet.amsdk.package` If you want to use a remote instance of the Client SDK, set the value of this property as follows:

```
com.iplanet.amsdk.package=remote
```

The default value is `ldap` if the property is not defined.

Federation Properties

The following properties are used to configure interactions in a federated environment. These properties are not automatically generated and placed in the `AMConfig.properties` file when you run the `make -f Makefile.clientsdk properties` command. You must manually add the properties to the file as needed.

`com.sun.identity.liberty.ws.soap.supportedActor`
Supported SOAP actors. Each actor must be separated by a pipe (|). Example:

```
com.sun.identity.liberty.ws.soap.supportedActors=
http://schemas.xmlsoap.org/soap/actor/next
```

`com.sun.identity.liberty.interaction.wspRedirectHandler`
Indicates the URL for `WSPRedirectHandlerServlet` to handle Liberty the WSF web service provider-resource owner. Interactions are based on user agent redirects. The servlet should be running in the same JVM where the Liberty service provider is running.

`com.sun.identity.liberty.interaction.wscSpecifiedInteractionChoice`
Indicates whether the web service client should participate in an interaction. Valid values are `interactIfNeeded | doNotInteract | doNotInteractForData` . Default value is `interactIfNeeded`. Default value is used if an invalid value is specified.

`com.sun.identity.liberty.interaction.wscWillInlcudeUserInteractionHeader`
Indicates whether the web service client should include `userInteractionHeader`. Valid values are `yes` and `no` (case ignored). Default value is `yes`. Default value is used if no value is specified.

`com.sun.identity.liberty.interaction.wscWillRedirect`
Indicates whether the web service client will redirect user for an interaction. Valid values are `yes` and `no`. Default value is `yes`. Default value is used if no value is specified.

`com.sun.identity.liberty.interaction.wscSpecifiedMaxInteractionTime`
Indicates the web service client preference for acceptable duration (in seconds) for an interaction. If the value is not specified or if a non-integer value is specified, then the default value is `60`.

`com.sun.identity.liberty.interaction.wscWillEnforceHttpsCheck`
Indicates whether the web service client enforces that redirected to URL is HTTPS. Valid values are `yes` and `no` (case ignored). The Liberty specification requires the value to be `yes`. Default value is `yes`. Default value is used if no value is specified.

`com.sun.identity.liberty.interaction.wspWillRedirect`
Indicates whether the web service provider redirects the user for an interaction. Valid values are `yes` and `no` (case ignored). Default value is `yes`. Default value is if no value is specified.

`com.sun.identity.liberty.interaction.wspWillRedirectForData`
Indicates whether the web service provider redirects the user for an interaction for data. Valid values are `yes` and `no`. Default value is `yes`. If no value is specified, the value is `yes`.

`com.sun.identity.liberty.interaction.wspRedirectTime`
Web service provider expected duration (in seconds) for an interaction. Default value if the value is not specified or is a non-integer value is `30`.

`com.sun.identity.liberty.interaction.wspWillEnforceHttpsCheck`
Indicates whether the web service client enforces that `returnToURL` is HTTP. Valid values are `yes` and `no` (case ignored). Liberty specification requires the value to be `yes`. Default value is `yes`. If no value is specified, then the value used is `yes`.

`com.sun.identity.liberty.interaction.wspWillEnforceReturnToHostEqualsRequestHost`
Indicates whether the web services client enforces that `returnToHost` and `requestHost` are the same. Valid values are `yes` and `no`. Liberty specification requires the value to be `yes`.

`com.sun.identity.liberty.interaction.htmlStyleSheetLocation`
Indicates the path to the style sheet used to render the interaction page in HTML.

`com.sun.identity.liberty.interaction.wmlStyleSheetLocation`
Indicates the path to the style sheet used to render the interaction page in WML.

Example:
`com.sun.identity.liberty.interaction.wmlStyleSheetLocation=/opt/SUNWam/lib/is-wml.xsl`

`com.sun.identity.liberty.ws.interaction.enable`
Default value is false.

`com.sun.identity.wss.provider.config.plugin=com.sun.identity.wss.provider.plugins.AgentPro`
Used by the web services provider to determine the plug-in that will be used to store the configuration.

Example:
`com.sun.identity.wss.provider.config.plugin=com.sun.identity.wss.provider.plugins.AgentI`

`com.sun.identity.loginurl`
Used by the web services clients in ClientSDK mode. Example:

`com.sun.identity.loginurl=https://`*hostName:portNumber*`/amserver/UI/Login`

`com.sun.identity.liberty.authnsvc.url`
Indicates the Liberty authentication service URL.

`com.sun.identity.liberty.wsf.version`
Used to determine which version of the Liberty identity web services framework is to be used when the framework can not determine from the inbound message or from the resource offering. This property is used when Access Manager is acting as the web service client. The default version is 1.1. The possible values are 1.0 or 1.1.

`com.sun.identity.liberty.ws.soap.certalias`
Value is set during installation. Client certificate alias that will be used in SSL connection for Liberty SOAP Binding.

`com.sun.identity.liberty.ws.soap.messageIDCacheCleanupInterval`
Default value is 60000. Specifies the number of milliseconds to elapse before cache cleanup events begin. Each message is stored in a cache with its `ownmessageID` to avoid duplicate messages. When a message's current time less the received time exceeds `thestaleTimeLimit` value, the message is removed from the cache.

`com.sun.identity.liberty.ws.soap.staleTimeLimit`
Default value is 300000. Determines if a message is stale and thus no longer trustworthy. If the message timestamp is earlier than the current timestamp by the specified number of milliseconds, the message the considered to be stale.

`com.sun.identity.liberty.ws.wsc.certalias`
Value is set during installation. Specifies default certificate alias for issuing web service security token for this web service client.

`com.sun.identity.liberty.ws.trustedca.certaliases`
Value is set during installation. Specifies certificate aliases for trusted CA. SAML or SAML BEARER token of incoming request. Message must be signed by a trusted CA in this list. The syntax is `cert alias 1[:issuer 1]|cert alias 2[:issuer 2]|.....` Example:

`myalias1:myissuer1|myalias2|myalias3:myissuer3`. The value issuer is used when the token doesn't have a KeyInfo inside the signature. The issuer of the token must be in this list, and the corresponding certificate alias will be used to verify the signature. If KeyInfo exists, the keystore must contain a certificate alias that matches the KeyInfo and the certificate alias must be in this list.

Using a Properties File

You can set properties in a properties file and then provide a path to it at runtime. The properties files must be in the CLASSPATH. The default properties file name is `AMConfig.properties` and is always read at start-up.

▼ To Set Client SDK Properties in a Properties File

1 **Generate a sample `AMConfig.properties` by running the following command:**

```
make -f Makefile.clientsdk properties
```

The `AMConfig.properties` will be present in the `temp` directory.

2 **Edit properties to suit your environment.**

Note – At runtime, if the file name is different from `AMConfig`, provide the file name (without the `.properties` extension) and path. The path should be in the CLASSPATH by declaring the JVM option:

`-Damconfig=`*filname*

Using the Java API

The ClientSDK properties can also be set programmatically using the class: `com.iplanet.am.util.SystemProperties`. See the following example.

EXAMPLE 1–1 Setting Client SDK Properties

```
import com.iplanet.am.util.SystemProperties;
import java.util.Properties;
public static void main(String[] args) {
        // To initialize a set of properties
        Properties props = new Properties();
        props.setProperty(”com.iplanet.am.naming.url’,
                    ”http://sample.com/amserver/namingservice’);
        props.setProperty(”com.sun.identity.agents.app.username’, ”amAdmin’);
```

EXAMPLE 1–1 Setting Client SDK Properties *(Continued)*

```
        props.setProperty("com.iplanet.am.service.password', "11111111');
        SystemProperties.initializeProperties(props) ;

        // To initialize a single property
        SystemProperties.initializeProperties("com.iplanet.am.naming.url',
                    "http://sample.com/amserver/namingservice');
    // Application specific code ...
}
```

Setting Individual Properties

You can set properties one at a time. For example, you can declare the following JVM option at run time to assign a value to a particular property:

`-D`*propertyName=propertyValue*

Setting Up a Client Identity

Some of the Access Manager components such as SAML, User Management, Policy, require an identity for the client. The client application reads configuration data to identify the client. You can set up the identity for the client in one of two ways:

- Set username and password properties can be authenticated
- Set an SSO Token Provider

Note – Some of the configuration attributes (such as password) are encrypted and stored in the data store as an Encryption/Decryption Key. If such attributes have to be decrypted by the client, the property must be set, and must be the same as that of the Access Manager Server.

This value is generated at installation time and stored in the following file:

Solaris	`/etc/opt/SUNWam/config/AMConfig.properties`
Linux	`/etc/opt/sun/identity/AMConfig.properties`
Windows	*AccessManager-base*`\identity\config\AMConfig.properties`
HP-UX	`/etc/opt/sun/identity/config/AMConfig.properties`

To Set Username and Password Properties

The following properties can be used to set the username and password that can be used by client SDK to obtain the configuration parameters. The authenticated username should have permissions to read the configuration data for SAML and User Management.

- The property to provide the user name is: `com.sun.identity.agents.app.username`
- The property to provide the plain text password is: `com.iplanet.am.service.password`

For scenarios where plain text password would be security concern, an encrypted password can be provided using the property: `com.iplanet.am.service.secret`.

If an encrypted password is provided, the encryption key must also be provided using the property: `am.encryption.pwd`.

To Set an SSO Token Provider

Set the following property: `com.sun.identity.security.AdminToken`

This provides an implementation for the interface, which returns the following single sign-on (SSO) token: `com.sun.identity.security.AppSSOTokenProvider`.

Building Custom Web Applications

The Client SDK package contains `Makefile.clientsdk` that you can use to generate and build samples and web applications. The makefile defines targets to build configuration properties, samples and web applications.

Building Stand-Alone Applications

Use these steps a template for building their identity-enabled web applications.

▼ To Build a Stand-Alone Application

1. **Install the Client SDK.**
   See "Installing the Client SDK" on page 14.
2. **Copy** `servlet.jar` **to** `lib` **directory.**

3 **Run the stand-alone application.**

Change directory to respective components within `clientsdk-samples`. Each has a `Readme.html` file explaining the changes to done and a Makefile to rebuild and run the program.

Targets Defined in clientsdk

For web deployment, `amclientwebapps.war` is ready to be deployed. However, you can make changes in `clientsdk-webapps` directory and the war file can be recreated.

Custom web applications can use the following as a template to build their identity enabled web application.

properties: Generates `AMConfig.properties` in the temp directory that can used as a template for setting AM SDK's properties

samples: Copies standalone samples and corresponding Makefiles to samples directory.

webapp: Generates `amclientwebapps.war` that can be deployed on any Servlet 2.3 compliant web container.

About the Client SDK Samples

Sample files are included in the Client SDK. These demonstrate how to write stand-alone programs and how to write web applications. After you deploy the Client SDK using either the Java ES installer or the `amconfig` script with `DEPLOY_LEVEL=9`, the Client SDK samples are available in the following directory:

Solaris	*AccessManager-base*`/SUNWam/war/`
Linux and HP-UX	*AccessManager-base*`/identity/war/`
Windows systems	*AccessManager-base*`\identity\war\`

The subdirectory `clientsdk-samples` includes samples for authentication, logging, policy and SAML stand-alone programs. The subdirectory`clientsdk-webapps` includes samples `for` user management, service management, and policy programs. Each sample has a `Readme.html` file with instructions on compiling and running the sample program.

Troubleshooting

If the client program receives a “New Generic Exception” message, one possible cause is that the session has expired. As a workaround, you can increase values of the Session Idle Timeout and Max Session Time attributes. In the Access Manager administration console, click Configuration > Global > Session. See "Maximum Session Time" and "Maximum Idle Time" in the section “Resulting Behavior If Session Quota Exhausted” in *Sun Java System Access Manager 7.1 Administration Reference* for attribute descriptions.

When attempting to authenticate against Access Manager, if the client program receives a “NullPointerException” message, one possible cause is that the client identity username, secret, or password properties are not set correctly. For more information, see “Security Credentials Properties” on page 21 and “Using the Java API” on page 27.

◆ ◆ ◆ CHAPTER 2

Using Authentication APIs and SPIs

This chapter provides information on using Sun Java™ System Access Manager 7.1 authentication programming interfaces to use and to extend the Authentication Service.

This chapter contains the following sections:

Overview of Authentication APIs and SPIs

Access Manager provides both Java APIs and C APIs for writing authentication clients that remote applications can use to gain access to the Authenticate Service. This communication between the APIs and the Authentication Service occurs by sending XML messages over `HTTP(S)`. The `remote-auth.dtd` is the template used in formatting the XML request messages sent to Access Manager and for parsing the XML return messages received by the external application. You can access `remote-auth.dtd` in the directory *AccessManager-base* `/SUNWam/dtd`.

New authentication modules are added to Access Manager by using the `com.iplanet.authentication.spi` package. The SPI implements the JAAS `LoginModule`, and provides additional methods to access the Authentication Service and module configuration properties files. Because of this architecture, any custom JAAS authentication module will work within the Authentication Service.

Note – If contacting the Authentication Service directly through its URL `http://`*AcceessManager-HostName.domain_name:port* `/`*service_deploy_uri*`/authservice` without the API, a detailed understanding of `remote-auth.dtd` will be needed for generating and interpreting the messages passed between the client and server.

How the Authentication Java APIs Work

External Java applications can authenticate users with the Access Manager Authentication Service by using the Authentication Java APIs. The APIs are organized in a package called `com.sun.identity.authentication` and can be executed locally or remotely. The classes and methods defined in this package are used to initiate the authentication process and communicate authentication credentials to the specific modules within the Authentication Service. The classes and methods can be incorporated into a Java application to allow communication with the Authentication Service.

The first step necessary for an external Java application to authenticate to Access Manager is to create a new `AuthContext` object (`com.sun.identity.authentication.AuthContext`). The `AuthContext` class is defined for each authentication request as it initiates the authentication process. Since Access Manager can handle multiple organizations, `AuthContext` is initialized, at the least, with the name of the organization to which the requestor is authenticating. Once an `AuthContext` object has been created, the `login()` method is called indicating to the server what method of authentication is desired.

`IndexName` is the value of the authentication type. The following table summarizes `IndexName` values and their corresponding authentication types.

TABLE 2–1 IndexName Values

`IndexName` **Value**	**Authentication Type**
`AuthContext.IndexType.ROLE`	Role-based
`AuthContext.IndexType.SERVICE`	Service-based
`AuthContext.IndexType.USER`	User-based
`AuthContext.IndexType.LEVEL`	Authentication Level-based
`AuthContext.IndexType.MODULE_INSTANCE`	Module-based

The `getRequirements()` method then calls the objects that will be populated by the user. Depending on the parameters passed with the instantiated `AuthContext` object and the two method calls, Access Manager responds to the client request with the correct login requirement screens. For example, if the requested user is authenticating to an organization configured for LDAP authentication only, the server will respond with the LDAP login requirement screen to

supply a user name and a password. The client must then loop by calling the `hasMoreRequirements()` method until the required credentials have been entered. Once entered, the credentials are submitted back to the server with the method call `submitRequirements()`. The final step is for the client to make a `getStatus()` method call to determine if the authentication was successful. If successful, the caller obtains a session token for the user; if not, a `LoginException` is thrown.

Because the Authentication Service is built on the JAAS framework, the Authentication API can also invoke any authentication modules written purely with the JAAS API.

For detailed information about Java APIs for authentication, see the Javadoc in the following directory:

AccessManager-base`/SUNWam/docs`

XML/HTTP Interface for Other Applications

Applications written in a programming language other than Java or C can exchange authentication information with Access Manager using the `XML/HTTP(s)` interface. Using the URL `http://`*server_name.domain_name* `:`*port*`/`*service_deploy_uri* `/authservice`, an application can open a connection using the HTTP POST method and exchange XML messages with the Authentication Service. The structure of the XML messages is defined in `remote-auth.dtd`. In order to access the Authentication Service in this manner, the client application must contain the following:

- A means of producing valid XML compliant with the `remote-auth.dtd`.
- HTTP 1.1 compliant client implementation to send XML-configured information to Access Manager.
- HTTP 1.1 compliant server implementation to receive XML-configured information from Access Manager.
- An XML parser to interpret the data received from Access Manager.

Examples of XML Messages

The following code examples illustrate how customers might configure the XML messages posted to the Authentication Service.

Note – Although the client application need only write XML based on the `remote-auth.dtd`, when these messages are sent they include additional XML code produced by the Authentication API. This additional XML code is not illustrated in the following examples.

The following example illustrates the initial XML message sent to the Access Manager. It opens a connection and asks for authentication requirements regarding the `exampleorg` organization to which the user will login.

EXAMPLE 2–1 Initial AuthContext XML Message

```
<?xml version="1.0" encoding="UTF-8"?>
<AuthContext version="1.0"><Request authIdentifier="0">
<Login orgName="dc=red,dc=iplanet,dc=com">
<IndexTypeNamePair indexType="moduleInstance"><IndexName>LDAP</IndexName>
</IndexTypeNamePair></Login></Request></AuthContext>
```

The following example illustrates the successful response from Access Manager that contains the `authIdentifier`, the session identifier for the initial request.

EXAMPLE 2–2 AuthIdentifier XML Message Response

```
<?xml version="1.0" encoding="UTF-8"?>
<AuthContext version="1.0"><Response authIdentifier="AQIC5wM2LY4SfczGP8Kp9
cqcaN1uW+C7CMdeR2afoN1ZxwY=@AAJTSQACMDE=#">
<GetRequirements><Callbacks length="3">
<PagePropertiesCallback isErrorState="false"><ModuleName>LDAP</ModuleName>
<HeaderValue>This server uses LDAP Authentication</HeaderValue>
<ImageName></ImageName><PageTimeOutValue>120</PageTimeOutValue>
<TemplateName></TemplateName>
<PageState>1</PageState>
</PagePropertiesCallback>
<NameCallback><Prompt> User Name: </Prompt></NameCallback>
<PasswordCallback echoPassword="false"><Prompt> Password: </Prompt>
</PasswordCallback></Callbacks></GetRequirements></Response></AuthContext>
```

The following example illustrates the client response message back to Access Manager. It specifies the type of authentication module needed by the user to log in.

EXAMPLE 2–3 Second Request Message With Authentication Module Specified

```
<?xml version="1.0" encoding="UTF-8"?>
<AuthContext version="1.0"><Request authIdentifier="AQIC5wM2LY4SfczGP8Kp9cqca
N1uW+C7CMdeR2afoN1ZxwY=@AAJTSQACMDE=#">
<SubmitRequirements><Callbacks length="2"><NameCallback><Prompt>User Name:</Prompt>
<Value>amadmin</Value>
</NameCallback>
<PasswordCallback echoPassword="false"><Prompt>Password:</Prompt>
<Value>admin123</Value>
</PasswordCallback></Callbacks></SubmitRequirements></Request></AuthContext>
```

The following example illustrates the return message from Access Manager which specifies the authentication module's login requirements.

EXAMPLE 2–4 Return XML Message With Login Callbacks

```
<?xml version="1.0" encoding="UTF-8"?>
<AuthContext version="1.0"><Response authIdentifier="AQIC5wM2LY4SfczGP8Kp9cqcaN1uW+
C7CMdeR2afoN1ZxwY=@AAJTSQACMDE=#">
<LoginStatus status="success" ssoToken="AQIC5wM2LY4SfczGP8Kp9cqcaN1uW+C7CMdeR2afoN1
ZxwY=@AAJTSQACMDE=#" successURL="http://blitz.red.iplanet.com/amserver/console">
<Subject>AQICOIy3FdTlJoAiOyyyZRTjOVBVWAb2e5MOAizI7ky3raaKypFE3e+GGZuX6chvLgDO32Zugn
pijo4xW4wUzyh2OAcdO9r9zhMU2Nhm206IuAmz9m18JWaYJpSHLqtBEcf1GbDrm3VAkERzIqsvkLKHmS1qc
yaT3BJ87wH0YQnPDze4/BroBZ8N5G3mPzPz5RbE07/1/w02yH9w0+UUFwwNBLayywGsr3bJ6emSSYqxos1N
1bo98xqL4FKAzItsfUAMd6v0ylWoqkoyoSdKYNHKbqvLDIeAfhqgoldxt64Or6HMXnOxz/jiVauh2mmwBpH
q1H2mOeF3agfUfuzKxBpLfELLwCH6QWcJmOZl0eNCFkGl7VwfnCJpTx1WcUhPSg0xD26D3dCQNruJpHPgzZ
FThe55M2gQ2qX+I1klmvzghSqiYfyoGg2SFeBeHE7iHuujO0e6UZgKDrOQPjU9aDh1GxxnsMQmaNkjuW+up
ghruWBGy+mDWmPQTme2bQWPIjBgB4wTDXTedeDzDBeulhCH4M0Ak9lvS7EIv6kHX5pRph6d0ND4/RVHka3k
WcQ5e0w2HpPjOxzNrWMfyXTkQJwOrA8yh1eBjG04VwiVqDV4wAV5EsIsIt0TrtAW2VZwV/KtLcGmjaKaT0H
dwRy0M4DHEqDbc6jF5ItVo9NneGFXMswPIoLm2nLuMrteAt7AtK7FGuCHlfYLavKoROtjaSuYTJGFwgz8Oi
vZ2r9boVnWVlz7ehwlyHvdfmpSKVl76Y4qEclX25m+lddAZE92RgSIrg97fp9gBOk2gVJWoQORNRDV2siHr
26 RiPLdvW3foG0hZgpLimJuLdByThRd/tdknDCCNRzelv7khr6nLPVPFVBgEJWlHmuffkdz4OsL0omFWpi
Jq05sQCPs/q6rq9ZJ98a8mcFK10BVPQki/1VfkIbKAdO4eswsIMalYkglBqXT4ARVTWRCWRNMCTDlQitF3g
T51AHn1WioFPm+NZ2KagVjQR6JFxHbdW0bKN7cLQViArJJFRtktR1BJh31/K+dAM2P+KbT1Lq13UUvXCynS
QwVbf7HJP5m3XrIQ6PtgZs4TB026H+iKy5T85YNL03j9sNnALiIKJEgvGLg2jxG+SU10xNLz3P3UVqmAnQI
9FIjmCtJcFtlLYR6BbkTvZVKxWz6+SoxNfDeKhIDwxkTNTLOzK491KzU/XAZTKmvdxTgf+WikbriBhFjsJ4
M6Npsq4p9Ksrjun9FVBTE/EUT5X/bY8zXLm0nw5KspQ7XRHPwrppQMVMMekz5qrNtQ9Cw/TeOhm4jvww/Bz
j4rydi7s7D10s2BWMfcuxmwQEipAWNmraKL37wWskrCdAzO2HXH4iJjWimiJ6J</Subject>
</LoginStatus></Response></AuthContext>
```

How the Authentication SPIs Work

Access Manager provides the capability to plug new, Java-based authentication modules into its framework allowing proprietary authentication providers to be managed using the Access Manager console. A custom authentication module must first be created using Java. Once created, the custom module can be added to the list of available authentication modules.

Note – This guide does not document the JAAS. For more information on these APIs, see the *Java Authentication And Authorization Service Developer's Guide*. Additional information can be found at `http://java.sun.com/products/jaas/`.

Extending the AMLoginModule Class

Custom authentication modules extend the `com.sun.identity.authentication.spi.AMLoginModule` class. The class must also implement the `init()`, `process()` and `getPrincipal()` methods in order to communicate

with the authentication module configuration files. The callbacks are then dynamically generated based on this file. Other methods that can be defined include `setLoginFailureURL` and `setLoginSuccessURL` which defines URLs to send the user to based on a failed or successful authentication, respectively.

Note – To make use of the account locking feature with custom authentication modules, the `InvalidPasswordException` exception should be thrown when the password is invalid.

Pluggable JAAS Module

The Java Authentication and Authorization Service (JAAS) is a set of APIs that enable services to authenticate and enforce access controls upon users. It implements a Java technology version of the standard Pluggable Authentication Module (PAM) framework, and supports user-based authorization. Access Manager supports pure JAAS pluggable authentication modules. In Access Manager, pure JAAS modules extend the JAAS `LoginModule` rather than `AMLoginModule`. A pure JAAS module is plugged in to the Authentication framework using the Authentication API.

Authentication Post Processing

The Authentication SPI includes the `AMPostAuthProcessInterface` which can be implemented for post-processing tasks. The following are examples of post-processing tasks:

- Adding attributes to a user's session after successful authentication
- Sending notification to an administrator after failed authentication
- General clean-up such as clearing cookies after logout or logging out of other system components.

The Core Authentication Service contains the Authentication PostProcessing Class attribute which contains the authentication post-processing class name as its value. Custom post processing interfaces can also be implemented.

`AMPostAuthProcessInterface` can be implemented for post authentication processing on authentication success, failure and logout. The SPI is configurable at the organization , service and role levels. The Authentication Service invokes the post processing SPI methods on successful, failed authentication and logout.

The `AMPostProcessInterface` class has 3 methods:

- "*onLoginSuccess*" on page 39
- "*onLoginFailure*" on page 39
- "*onLogout*" on page 39

Some supporting information on these methods is provided in the following sections. For a comprehensive listing and detailed information on all Access Manager methods, see the Javadoc installed in the following directory:

```
AccessManager-base/SUNWam/docs
```

onLoginSuccess

This method should be implemented for post-processing after a successful authentication. Authentication Service will invoke this method on successful authentication.

Method signature is:

```
public void onLoginSuccess(Map requestParamsMap,
                           HttpServletRequest request,
                           HttpServletResponse response,
                           SSOToken ssoToken)
    throws AuthenticationException;
```

where

- `requestParamaMap` is a map containing `HttpServletRequest` parameters
- request `HttpServletRequest object`
- `response HttpServletResponse object`

  `com.sun.identity.authentication.spi.AuthenticationException` is thrown on error.

onLoginFailure

This method should be implemented for post processing after a failed authentication. Authentication Service will invoke this method on failed authentication.

Method signature is:

```
public void onLoginFailure(Map requestParamsMap,
                           HttpServletRequest request,
                           HttpServletResponse response)
   throws AuthenticationException;
```

where

- `requestMap` is a map containing `HttpServletRequest` parameters
- request `HttpServletRequest` object
- response `HttpServletRequest` object

  `com.sun.identity.authentication.spi.AuthenticationException` is thrown on error.

onLogout

This method should be implemented for post-processing on a logout request. Authentication Service will invoke this method on logout.

Method signature is:

```
public void onLogout(HttpServletRequest request,
                     HttpServletResponse response,
                    SSOToken ssoToken)
   throws AuthenticationException;
```

where

- request `HttpServletRequest` object is a map containing `HttpServletRequest` parameters
- response `HttpServletResponse` object
- `ssoToken` authenticated user's single sign on token

`com.sun.identity.authentication.spi` `AuthenticationException` is thrown on error.

Using Authentication APIs

Access Manager comes with a number of sample programs that demonstrate how you can use the Authentication APIs to extend the functionality of the authentication service and authentication modules.

Running the Sample Authentication Programs

The source code and Makefile are provided for all sample programs. For some sample programs, additional supporting files are also included. The instructions for compiling and executing the sample programs are the same for all samples described in this section.

Java API Code Samples and Their Locations

The following tables describe the locations of all the files you need to implement the sample programs on various platforms, and the variable names used for default directories in the source code and Makefiles. Table 2–2 summarizes file locations and variable names used for Solaris Sparc/x86.l Table 2–3 summarizes default directories for Linux. Table 2–4 summarizes default directories for Windows 2000.

TABLE 2–2 Default directories for Solaris Sparc/x86

Variable	Description	Location
Api_sample_dir	Directory that contains authentication API sample files	`<install_root>/SUNWam/ samples/authenitcation/api`
Config_directory	Directory that contains configuration files	`/etc/opt/SUNWam/config`
Product_Directory	Directory where Access Manager is installed.	`install_root>/SUNWam`

TABLE 2–3 Default directories for Linux

Variable	Description	Location
Api_Sample_Dir	Directory that contains authentication API sample files	`<install_root>/sun/ identity/samples/authentication/api`
Config_Directory	Directory that contains configuration files	`/etc/opt/sun/identity/config`
Product_Directory	Directory where Access Manager is installed.	`<install_root>/sun/identity`

TABLE 2–4 Default directories for Windows 2000

Variable	Description	Location
Api_Sample_Dir	Directory that contains authentication API sample files	`<install_root>\samples\ authentication\api`
Config_Directory	Directory that contains configuration files	`<install_root>\lib`
Product_Directory	Directory where Access Manager is installed.	`<install_root>`

These steps are for all platforms.

▼ To Compile and Execute the Java API Samples

1 In the Makefile, modify the following variables as necessary to suit your Access Manager installation:

BASE_DIR: Enter the path to the directory where Access Manager is installed.

JAVA_HOME: Enter the path to the directory where the Java compiler is installed.

DOMAIN: Enter the name of the organization to login to.

SHARE_LIB: Enter the path to the directory where Access Manager jar files are stored.

JSS_JAR_PATH: Enter the path to the directory where JSS jar files are stored.

JSSPATH: Enter the path to the directory where JSS libraries are located.

2 **In the Certificate Sample Makefile only, modify the following as necessary:**

CERTNICKNAME: Enter the Certificate nickname.

URL: Enter the Access Manger Server URL.

PASSWORD: Enter the Certificate DB Password.

3 **Copy** `AMConfig.properties` **from** *Config_Directory* **in the Access Manager server installation to the client machine.**

(Note: For SSL check SSL Configuration Setup, step 2).

4 **In the Makefile, update the classpath to include the location of the newly created** `AMConfig.properties`**.**

5 **In the client machine, create a directory named** `locale`**.**

Copy all the property files from the `locale` directory in the Access Manager server installation machine to the client machine. The `locale` directory on the server machine can be found under the *Product_Directory*.

6 **Update the classpath in the Makefile to include the location of newly created** `locale` **files.**

7 **Include** `jaas.jar` **in your classpath if you are using a JDK version less than JDK1.4**

8 **Compile the program.**

- On Solaris Sparc/x86, Linux, run the `gmake` command.
- On Windows 2000, run the `make` command.

9 **Run the sample program.**

- On Solaris Sparc/x86 or Linux, run the following command: `gmake run`
- On Windows 2000, run the following command: `make run`

▼ To Configure SSL for Java API Samples

1. **In the Makefile, add this JVM property in the `run` target:**

   `-D "java.protocol.handler.pkgs=com.iplanet.services.comm"`

2. **Copy `AMConfig.properties` from *Config_Directory* in the Access Manager server installation to the client machine.**

3. **Edit the following properties in `AMConfig.properties`.**

   com.iplanet.am.admin.cli.certdb.dir: Enter the path to the certificate database directory.

   com.iplanet.am.admin.cli.certdb.prefix: Enter the certificate database prefix.

4. **In the LDAP and JCDI Samples only:**

   com.iplanet.am.server.protocol: Change the value to HTTPS.

   com.iplanet.am.server.port: Enter the appropriate port number from the server machine.

5. **Create or copy the certificate database file to the certificate db directory. Use the directory name in `com.iplanet.am.admin.cli.certdb.dir`.**

6. **Rename the file to use the prefix specified in the property `com.iplanet.am.admin.cli.certdb.prefix`.**

   For the details, see the Javadoc for the Remote Client API.

LDAPLogin Example

The `LDAPLogin` sample is an example of a custom Java application that uses the authentication remote APIs to authenticate to the LDAP module. You can modify the sample source code to authenticate to other existing or customized authentication modules. The sample source code, Makefile, and `Readme.html` are located in the following directory:

AccessManager-base`/ SUNWam/samples/authentication/LDAP`

To compile and run the sample program, follow the steps in "To Compile and Execute the Java API Samples" on page 41.

CertLogin Example

The CertLogin sample is an example of a custom Java application that uses digital certificates for authentication. You can modify the sample source code to authenticate to other existing or customized authentication modules. The sample source code, `Makefile`, and `Readme.html` are located in the following file:

AccessManager-base/ SUNWam/samples/authentication/Cert

▼ To Run the CertLogin Program

1. **Enable SSL.**
   Follow the instructions in "To Configure SSL for Java API Samples" on page 43.

2. **Compile and execute the sample code.**
   See "To Compile and Execute the Java API Samples" on page 41

Using certutil for Client Certificate Management

`Certutil` is a command-line utility that can create and modify cert7.db and key3.db database files. It can also list, generate, modify, or delete certificates within the cert7.db file and create or change the password, generate new public and private key pairs, display the contents of the key database, or delete key pairs within the key3.db file. The key and certificate management process usually begins with creating keys in the key database, then generating and managing certificates in the certificate database.

JCDI Module Example

The JCDI Module Example demonstrates the use of Java Card Digital ID (JCDI) authentication with Access Manager. The sample has two components:

- Remote client
- Server JCDI authentication module

The remote client component is located in the following directory:

AccessManager-base/`samples/authentication/api/jcdi`

The server JCDI authentication module is located in the following directory:

AccessManager-base`samples/authentication/spi/jcdi`

The sample illustrates JCDI authentication using the Remote Authentication API. You can modify the sample source code to authenticate to other existing or customized authentication modules. The source code, `Makefile`, and `Readme.html` are located in the following directory:

AccessManager-base`samples/authentication/api/jcdi`

To compile and run the sample program, follow the steps in "Running the Sample Authentication Programs" on page 40.

Using Authentication SPIs

Access Manager provides the following sample programs to demonstrate how you can use the Authentication service programming interfaces (SPIs) to extend authentication functionality:

Implementing a Custom Authentication Module

Access Manager contains a sample exercise for integrating a custom authentication module with files that have already been created. This sample illustrates the steps for integrating an authentication module into the Access Manager deployment. All the files needed to compile, deploy and run the sample authentication module can be found in the following directory:

AccessManager-base`/SUNWam/samples/authentication/providers`

The following sections will use files from this sample as example code:

About the Login Module Sample

<PRODUCT_DIR> setting on different Platforms:

Solaris Sparc/x86: <PRODUCT_DIR> = *base-directory*`/SUNWam`

Linux: <PRODUCT_DIR> = *base-directory*`/sun/identity`

Windows 2000: <PRODUCT_DIR> = *base-directory*

Writing a Sample Login Module

Use the `AMLoginModule` SPI to write your own sample login module.

▼ To Write a Sample Login Module

1 **"Creating a Module Properties File" on page 46.**

2 **"Writing the Principal Class" on page 47.**

3 **"Implementing the LoginModule Interface" on page 47.**

The following are the default directories used in the sample exercise for the various platforms:

Solaris Sparc/x86: <PRODUCT_DIR> = *base-directory*`/SUNWam`

Linux: <PRODUCT_DIR> = *base-directory*/sun/identity

W2K: <PRODUCT_DIR> = *base-directory*

Creating a Module Properties File

Create a Module properties XML file with the same name of the class (no package name) and use the extension `.xml`. You must create an XML file with this naming convention even if no states required

Based on this configuration file, the Authentication user interface will dynamically generate a login page.

You can define page states in the module properties file as shown in "Creating a Module Properties File" on page 46. Each callback element corresponds to one login page state. When an authentication process is invoked, `Callback[]` values will be generated from the user's Login Module for each state. All login state definitions start with 1. The module controls the login process, and then determines what the next state is.

`Auth_Module_Properties.dtd` defines the data structure that will be used by each authentication module to specify its properties. `Auth_Module_Properties.dtd` provides definitions to initiate, construct and send required callbacks information to the Authentication graphical user interface. `Auth_Module_Properties.dtd` is stored in the <PRODUCT_DIR>`/dtd` directory.

EXAMPLE 2–5 Module Configuration Sample

```
<ModuleProperties moduleName="LoginModuleSample" version="1.0" >
         <Callbacks length="2" order="1" timeout="60"
                           header="This is a sample login page">
             <NameCallback>
             <Prompt> User Name </Prompt>
         </NameCallback>
         <NameCallback>
             <Prompt> Last Name </Prompt>
         </NameCallback>
```

EXAMPLE 2–5 Module Configuration Sample *(Continued)*

```
        </Callbacks>
        <Callbacks length="1" order="2" timeout="60"
                          header="You made it to page 2" >
            <PasswordCallback echoPassword="false" >
            <Prompt> Just enter any password </Prompt>
            </PasswordCallback>
        </Callbacks>
    </ModuleProperties>
Module Configuration Sample
```

In this module configuration sample, `page state` one has two callbacks. The first callback is for user ID, and second is for Last Name. When the user fills in the callbacks, the following events occur:

- The `Callback[]` values are sent to the module.
- The `process()` routine validates the callback values.
- The module writer sets the next page state to 2.

  Page state 2 has one callback to request the user to enter a password. The `process()` routine is again called after the user submits the `Callback[] values.` If the module writer throws a `LoginException`, then an Authentication Failed page will be sent to the user. If no exception is thrown, the user is redirected to his or her default page.

Writing the Principal Class

After creating module configuration XML file, the next step is to write a Sample Principal class which implements `java.security.Principal`. The constructor takes the user's username as an argument. If authentication is successful, the module will return this principal to Authentication framework. The Authentication framework populates a Subject with a `SamplePrincipal` representing the user.

Implementing the LoginModule Interface

`AMLoginModule` is an abstract class which implements `JAAS LoginModule`. `AMLoginModule` provides methods for accessing Access Manager services and the module XML configuration. `Login Module` writers must subclass `AMLoginModule` class and implement the following methods:

- `init()`
- `process()`
- `getPrincipal()`

For detailed descriptions, syntax, and parameters, see the *Sun Java System Access Manager 7.1 Java API Reference*. The following sections provide some supporting information about these methods.

init() This is an abstract method, Module writer should implement to initialize this LoginModule with the relevant information. If this `LoginModule` does not understand any of the data stored in `sharedState` or options parameters, the data can be ignored. This method is called by a `AMLoginModule` after this`SampleLoginModule` has been instantiated, and prior to any calls to its other public methods. The method implementation should store away the provided arguments for future use. The init method may additionally peruse the provided `sharedState` to determine what additional authentication state it was provided by other `LoginModules`, and may also traverse through the provided options to determine what configuration options were specified to affect the LoginModule's behavior. It may save option values in variables for future use.

process() The process method is called to authenticate a Subject. This method implementation should perform the actual authentication. For example, it may cause prompting for a user name and password, and then attempt to verify the password against a password database. If your LoginModule requires some form of user interaction (retrieving a user name and password, for example), it should not do so directly. That is because there are various ways of communicating with a user, and it is desirable for `LoginModules` to remain independent of the different types of user interaction. Rather, the LoginModule's process method should invoke the handle method of the `CallbackHandler` passed to this method to perform the user interaction and set appropriate results, such as the user name and password and the `AMLoginModule` internally passes the GUI an array of appropriate Callbacks, for example a `NameCallback` for the user name and a `PasswordCallback` for the password, and the GUI performs the requested user interaction and sets appropriate values in the Callbacks.

Consider the following points while writing the `process()` method:

- Perform the authentication. If Authentication succeeded, save the principal who has successfully authenticated.
- Return -1 if authentication succeeds, or throw a `LoginException` such as `AuthLoginException` if authentication fails or return relevant state specified in module configuration XML file
- If multiple states are available to the user, the Callback array from a previous state may be retrieved by using the `getCallbak(int state)` methods. The underlying login module keeps the `Callback[]` from the previous states until the login process is completed.
- If a module writer needs to substitute dynamic text in next state, the writer could use the `getCallback()` method to get the `Callback[]` for the next state, modify the output text or prompt, then call `replaceCallback()` to update the Callback array. This allows a module writer to dynamically generate challenges, passwords or user IDs. Each authentication session will create a new instance of your Login Module Java class. The reference to the class

will be released once the authentication session has either succeeded or failed. It is important to note that any static data or reference to any static data in your Login module must be thread-safe.

getPrincipal() This method should be called once at the end of a successful authentication session. A login session is deemed successful when all pages in the Module properties XML file have been sent and the module has not thrown an exception. The method retrieves the authenticated token string that the authenticated user will be known by in the Access Manager environment.

Note – If the custom authentication module requires or already uses a service configuration XML file:

- The XML file should contain attribute schema for one of the following attributes: `iplanet-am-auth-`*authModuleName*`-auth-level` or `lsunAMAuth`*authModuleName*`AuthLevel`
- The module Java file should invoke the following method in the `init` method implementation: `public boolean setAuthLevel(int auth_level)`

Compiling and Deploying the LoginModule program

If you are writing your own Custom Authentication module based on the `AMLoginModule` SPI or a pure JAAS module, then you can skip this step. Otherwise, after writing the sample Login Module, compile and deploy the sample found under *AccessManager-base*`/samples/authentication/spi/providers`.

▼ To compile the Login Module

1 Set the following environment variables.

These variables will be used to run the `gmake` command. You can also set these variables in the Makefile. This Makefile is in the following directory: *AccessManager-base* `/samples/authentication/spi/providers`.

JAVA_HOME: Set this variable to your installation of JDK. The JDK should be version 1.3.1_06 or higher.

CLASSPATH: Set this variable to refer to `am_services.jar` which can be found in the *Idetnity_base*`/lib directory`. Include `jaas.jar` in your classpath if you are using JDK version less than JDK1.4

BASE_DIR: Set this variable to the directory where the Access Manager is installed.

BASE_CLASS_DIR: Set this variable to the directory where all the Sample compiled classes are located.

JAR_DIR: Set this variable to the directory where the JAR files of the Sample compiled classes will be created.

2 **In the** *AccessManager-base*`/samples/authentication/spi/providers` **directory, run** `gmake`**.**

▼ To Deploy the Login Module Sample Program

1 **Copy** `LoginModuleSample.jar` **from** *JAR_DIR* **to** *AccessManager-base*`/web-src/services/WEB-INF/lib.`

2 **Copy LoginModuleSample.xml from** *AccessManager-base* `/samples/authentication/spi/providers` **to** *AccessManager-base* `/web-src/services/config/auth/default.`

3 **Redeploy the** `amserver.war` **file.**

▼ To Redeploy the amserver.war File

1 **In** *AccessManager-base*`/bin/amsamplesilent`**, set Deploy Level variable as follows:**

DEPLOY_LEVEL=21

2 **In** *AccessManager-base*`/bin/amsamplesilent`**, set container-related environment variables.**

- On Sun Java System Web Server 6.1, where `/amserver` is the default `DEPLOY_URI`:

  ```
  SERVER_HOST=WebServer-hostName
  SERVER_PORT=WebServer-portNumber
  SERVER_PROTOCOL=[http | https]
  SERVER_DEPLOY_URI=/amserver
  WEB_CONTAINER=WS6
  WS61_INSTANCE=https-$SERVER_HOST
  WS61_HOME= WebServer-base-directory
  WS61_PROTOCOL=$SERVER_PROTOCOL
  WS61_HOST=$SERVER_HOST
  WS61_PORT=$SERVER_PORT
  WS61_ADMINPORT=WebServer-adminPortWS61_ADMIN=WebServer-adminUserName
  ```

- On Sun Java System Application Server 7.0, where `/amserver` is the default `DEPLOY_URI`:

  ```
  SERVER_HOST=ApplicationServer-hostName
  SERVER_PORT=ApplicationServer-portNumber
  SERVER_PROTOCOL=[http | https]
  SERVER_DEPLOY_URI=/amserver
  WEB_CONTAINER=AS7
  AS70_HOME=/opt/SUNWappserver7
  ```

```
AS70_PROTOCOL=$SERVER_PROTOCOL
AS70_HOST=$SERVER_HOST
AS70_PORT=$SERVER_PORT
AS70_ADMINPORT=4848
AS70_ADMIN=admin
AS70_ADMINPASSWD=ApplicationServer-adminPassword
AS70_INSTANCE=server1
AS70_DOMAIN=domain1
AS70_INSTANCE_DIR=/var/opt/SUNWappserver7/domains/
  ${AS70_DOMAIN:-domain1}/${AS70_INSTANCE:-server1}
AS70_DOCS_DIR=/var/opt/SUNWappserver7/domains/${AS70_DOMAIN:-domain1}/
  ${AS70_INSTANCE:-server1}/docroot
#If Application Server is SSL Enabled then set the following:
#AS70_IS_SECURE=true
#SSL_PASSWORD=SSLpassword
```

- On other supported platforms:

  Set platform-specific variables as is appropriate for the container.

3 Redeploy the services web application by running the following command:

AccessManager-base`/bin/amconfig -s`
AccessManager-base`/bin/amsamplesilent`

4 Restart the container instance.

- Web Server example:

  /WebServer-base-directory/
  `https-`*WebServer-instanceName*`/restart`

- Application Server example:

```
/var/opt/SUNWappserver7/domains/${AS70_DOMAIN:-domain1}/
  ${AS70_INSTANCE:-server1}/bin/restartserv
```

Loading the Login Module Sample into Access Manager

Once you've compiled and deployed the login module, you must load the login module into Access Manager. You can load the login module by using either the Access Manager administration console, or by using the `amadmin` command.

▼ To Load the Login Module Using the Administration Console

1 Login to Access Manager Console as amadmin, using the URL:

`http://`*host.domain:port*`/`*Console-Deploy-URL*

2 **Click Configuration.**

3 **In the Configuration tab, under Authentication, click Core.**

4 **Add class file name** `com.iplanet.am.samples.authentication.spi.providers.LoginModuleSample` **to the Pluggable Authentication Modules Classes list.**

5 **Click Save.**

▼ To Load the Login Module Using the Command Line

1 **Write a sample XML file as shown in "To Load the Login Module Using the Command Line" on page 52, which will add the** `LoginModuleSample` **authentication module entry into the allowed modules and an authenticators list.**

```
<!--
      Copyright (c) 2003 Sun Microsystems, Inc.
               All rights reserved
     Use is subject to license terms.
-->

 <!DOCTYPE Requests
      PUBLIC "-//iPlanet//iDSAME 5.0 Admin CLI DTD//EN"
     "jar://com/iplanet/am/admin/cli/amAdmin.dtd"
 >

<Requests>

    <SchemaRequests serviceName="iPlanetAMAuthService"
                      SchemaType="Global">
      <AddDefaultValues>
       <AttributeValuePair>
          <Attribute name="iplanet-am-auth-authenticators"/>

 <Value>com.iplanet.am.samples.authentication.spi.providers.
                      LoginModuleSample</Value>
        </AttributeValuePair>
      </AddDefaultValues>

    </SchemaRequests>
</Requests>
```

2 **Use** `amadmin` **to load** `sample.xml`**:**

```
<AMADMIN> --runasdn uid=amAdmin,ou=People,<root_suffix> --password <password>
--data sample.xml
```

Solaris Sparc/x86: `AMADMIN = <PRODUCT_DIR>/bin/amadmin`

On W2K: `AMADMIN = <PRODUCT_DIR>\\bin\\amadmin`

Running the LoginModule Sample Program

This sections provides instructions for running the login module on Solaris and on Windows platforms.

▼ To Run the LoginModule on Solaris

1. **Use the following URL to log in to Access Manager console as `amAdmin`:**

   `http://`*host.domain:port*`/`*Console-Deploy-URI*

2. **Click Identity Management, and in the Identity Management view select your organization.**

3. **From the View menu, select Services.**

4. **In the navigation frame, under Authentication, click Core.**

5. **Select`LoginModuleSample` to add it to the list of highlighted modules in Organization Authentication Modules.**

   Make sure LDAP module is also selected. If not selected, you will not be able to login to Access Manager Console. You can use Control + mouse click to add additional modules.

6. **Click Save.**

7. **Log out.**

8. **Enter the following URL:**

   `http://`*host.domain:port*`/`*Service-Deploy-URI*`/UI/Login?module=LoginModuleSample`

   If you choose to use an organization other than the default, be sure to specify that in the URL using the `org` parameter.

▼ To Run the Login Module on Windows 2000

1. **Set the following environment variables. These variables will be used to run the make command. You can also set these variables in the Makefile.**

   This Makefile is in the same directory as the Login Module Sample program files: *AccessManager-base*`\samples\authentication\spi\providers`

   JAVA_HOME: Set this variable to your installation of JDK. The JDK should be version 1.3.1_06 or higher.

BASE: Set this variable to *base-directory*

CLASSPATH: Set this variable to refer to `am_services.jar` which can be found in the *base-directory*`\lib` `directory`. Include `jaas.jar` in your classpath if you are using JDK version less than JDK1.4

BASE_CLASS_DIR: Set this variable to the directory where all the Sample compiled classes are located.

JAR_DIR: Set this variable to the directory where the JAR files of the Sample compiled classes will be created.

2 **In the** *base-directory*`\samples\authentication\spi\providers` **directory, run the** `make` **command.**

▼ To Deploy the Login Module

1 **Copy** `LoginModuleSample.jar` **from** `JAR_DIR` **to** *AccessManager-base*`\web-src\services\WEB-INF\lib`

2 **In the Web Container from which this sample has to run, update the classpath with** `LoginModuleSample.jar`**.**

3 **Update** `server.xml` **with the new classpath and** `server.xml` **locations:**

- Sun Java System Web Server : `<WS-install-dir>\https-<WS-instance-name>\config\server.xml`
- Sun Java System Application Server: `<AS-install-dir>\domain<appserver domain><appserver_instance>\config\server.xml`

  Example:`<AS-install-dir>\domain\domain1\server1\config\server.xml`

4 **Copy** `LoginModuleSample.xml` **from** *base-directory* `\samples\authentication\spi\providers` **to** *base-directory*`\web-src\services\config\auth\default`**.**

5 **Restart the web container**

Web Server: `<WS-home-dir>\https-<WS-instance-name>\restart`

Application Server: `AppServer-home-dir>\domains\<domain name><server_instance>\bin\restartserv`

Implementing Authentication PostProcessing SPI

The Authentication SPI includes the `AMPostAuthProcessInterface` which can be implemented for post-processing tasks. The `AMPostProcessInterface` Javadoc are available at:

AccessManager-base/SUNWam/docs/com/sun/identity/authentication/spi/AMPostAuthProcessInterface.html

The SPI is configurable at the organization, service and role levels. The Authentication Service invokes the post processing SPI methods on successful or failed authentication and on logout.

About the PostProcessing SPI Sample

<PRODUCT_DIR> or *AccessManager-base* directory on different Platforms:

- Solaris Sparc/x86: *AccessManager-base*/SUNWam
- Linux: *AccessManager-base*/sun/identity

▼ To Compile the ISAuthPostProcessSample Program on Solaris Sparc/x86 or Linux

Follow these steps given below to compile the sample found under *AccessManager-base*/samples/authentication/spi/postprocess.

1 **Set the following environment variables.**

JAVA_HOME: Set this variable to your installation of JDK. The JDK should be version 1.3.1_06 or higher.

CLASSPATH: Set this variable to refer to am_services.jar which can be found in the *AccessManager-base*/lib directory. Include jaas.jar in your classpath if you are using JDK version lower than JDK1.4

BASE_DIR: Set this variable to the directory where Access Manager is installed.

BASE_CLASS_DIR: Set this variable to the directory where all the Sample compiled classes are located.

JAR_DIR: Set this variable to the directory where the JAR files of the Sample compiled classes will be created.

These variables will be used to run the gmake command. You can also set these variables in the Makefile. This Makefile is in the following directory: *AccessManager-base*/samples/authentication/spi/postprocess.

2 **In the directory** *AccessManager-base* /samples/authentication/spi/postprocess, **run the** gmake **command.**

▼ To Deploy the ISAuthPostProcess Sample Program

1. **Copy** `ISAuthPostProcess.jar` **from** `JAR_DIR` **to** *AccessManager-base*`/lib`.

2. **Update the Web Container configuration file** `server.xml`.

   Add `ISAuthPostProcessSample.jar` to the classpath. The `server.xml` file for different web containers can be found at the following locations:

   Web Server: `<WS-home-dir>/https-<WS-instance-name>/config/`

   Application Server:`<AS-home-dir>/domain/domain1/server1/config/`

   For all other web containers consult, the manufacturer's documentation.

3. **Restart the web container.**

   Web Server: `<WS-home-dir>/https-<WS-instance-name>/restart`

   Application Server: `<AS-install-dir>/<domains>/<domain name>/<server instance>/bin/restartserv` Example: `/<AS-home-dir>/domains/domain1/server1/bin/restartserv`

   For all other web containers consult their documentation.

Configuring the Authentication Post Processing SPI

The Authentication PostProcessing Sample can be configured at the Organization, Service or Role level.

▼ To Configure ISAuthPostProcess Sample for an Organization

1. **Log in to Access Manager console as** `amAdmin`**. Use the following URL:**

   `http://`*host.domain:port*`/`*Console-Deploy-URI*

2. **Click Identity Management, and select your organization.**

3. **From the View menu, click Services.**

4. **In the navigation frame, under Authentication, click Core.**

5. **Add the following to the Authentication PostProcessing Class attribute:**

   `com.iplanet.am.samples.authentication.spi.postprocess`

6. **Add the following to the** `Authentication PostProcessing Class` **attribute:**

   `ISAuthPostProcessSample`

7. **Click Save.**
8. **Log out.**
9. **Go to the following URL**

   If you choose to use an organization other than the default, be sure to specify that in the URL using the `org` parameter.

   The postprocessing SPI will be executed on successful authentication, on failed authentication, and on Logout.

▼ To Configure the ISAuthPostProcess Sample for a Service

1. **Log in to Access Manager console as `amAdmin`. Use the following URL:**

   `http://<host>.<domain>:<port>/<Console-Deploy-URI>`
2. **Click Identity Management, and select your organization.**
3. **From the View menu, select Services.**
4. **Select Authentication Configuration**
5. **From the Service Instance frame, select New Instance.**
6. **Enter a name for the service.**
7. **Add the following to the Authentication PostProcessing Class attribute:**
   `com.iplanet.am.samples.authentication.spi.postprocess. ISAuthPostProcessSampl`
8. **Click Submit to save the changes.**
9. **Click Service Name and define the Authentication Configuration for the new service.**
10. **Log out.**
11. **Go to the following URL:** `http://`*host.domain:port/Service-Deploy-URI*`/UI/Login?service=`*servicename*

    If you choose to use an organization other than the default, be sure to specify that in the URL using the `org` parameter.

    The postprocessing SPI will get executed on successful authentication, failed authentication and on Logout for the service accessed.

▼ To Configure ISAuthPostProcess Sample for a Role

1. **Log in to Access Manager console as** `amAdmin`**. Use the following URL:**
   `http://`*host.domain:port/Console-Deploy-URI*
2. **Click the Identity Management tab, and select your organization.**
3. **From the View menu, select Roles to view the role properties.**
4. **From the View menu, select Services.**
5. **Click Edit to edit the authentication configuration.**
6. **Add the following to the Authentication post Processing Class attribute:**
   `com.iplanet.am.samples.authentication.spi.postprocess. ISAuthPostProcessSample`
7. **Click Submit to save the changes.**
8. **Log out.**
9. **Go to the following URL:**
   `http://`*host.domain:port*/*Service-Deploy-URI*`/UI/Login?role=`*roleName*

   If you choose to use an organization other than the default, be sure to specify that in the URL using the `org` parameter. Example: `org=`*orgName*

   The postprocessing SPI will be executed for the service accessed on successful authentication, on failed authentication, and on Logout.

Compiling On Windows 2000

Go to the *base-directory*`\samples\authentication\spi\postprocess` directory and run the `make` command.

▼ To Deploy the ISAuthPostProcessSample Program

1. **Copy** `ISAuthPostProcess.jar` **from** `JAR_DIR` **to** *base-directory*`\lib`
2. **In the Web Container from which this sample has to run, update the classpath with** `ISAuthPostProcess.jar`**.**
3. **Restart Access Manager.**
   base-directory`\bin\amserver start`

To Configure Authentication Post Processing SPI

This sample can be can be set in the Core Authentication Service for Organization and Authentication Configuration Service for Role OR Service.

See the section "Configuring the Authentication Post Processing SPI" on page 56.

Generating an Authentication User ID

This file explains how to compile, deploy and configure the Authentication User ID Generation SPI Sample.

In the following sections, the PRODUCT_DIR setting depends on which platform you're using:

Solaris Sparc/x86: `PRODUCT_DIR = <install_root>/SUNWam`

Linux: `PRODUCT_DIR = <install_root>/sun/identity`

▼ To Compile the UserIDGeneratorSample on Solaris Sparc/x86, Linux

The sample is located in the following directory:

AccessManager-base`/samples/authentication/spi/genuid`

1 Set the following environment variables.

These variables will be used to run the `gmake` command. You can also set these variables in the Makefile which is located in the following directory:

AccessManager-base`/samples/authentication/spi/genuid`

JAVA_HOME: Set this variable to your installation of JDK. The JDK should be version 1.3.1_06 or higher.

CLASSPATH: Set this variable to refer to `am_services.jar` which can be found in the `<PRODUCT_DIR>/lib directory`. Include `jaas.jar` in your classpath if you are using JDK version less than JDK1.4.

BASE_DIR: Set this variable to the directory where the Access Manager is installed.

BASE_CLASS_DIR: Set this variable to the directory where all the Sample compiled classes are located.

JAR_DIR: Set this variable to the directory where the JAR files of the Sample compiled classes will be created.

2 **In the directory** *AccessManager-base* `/samples/authentication/spi/genuid`, **run the** `gmake` **command:**

▼ To Deploy the UserIDGeneratorSample Program

1 **Copy** `UserIDGeneratorSample.jar` **from** `JAR_DIR` **to** *AccessManager-base*`/lib`**.**

2 **in the Web Container from which this sample has to run, update the classpath with** `UserIDGeneratorSample.jar`**.**

- On Sun ONE Web Server, go to server instance `configurationdirectory`:

  `<WS-home-dir>/https-<WS-instance-name>/config/`
- On Sun ONE Application Server, in the directory `<AS-home-dir>/domain/domain1/server1/config/` update `server.xml` with the new classpath.
- For all other containers, consult the documentation that came with the product.

3 **Restart web container.**`<WS-home-dir>/https-<WS-instance-name>/start`

`<AS-home-dir>/domains/domain1/server1/bin/start`

Configuring the UserIDGeneratorSample Program

The Authentication User ID Generation Sample can be configured at the Organization level, and then used or invoked by the out-of-box Membership/Self- registration authentication module.

▼ To Configure UserIDGeneratorSample for an Organization

1 **Log in to Access Manager console as** `amAdmin`**. Use the following URL:**

`http://`*host.domain:port/Console-Deploy-URI*

2 **Click the Identity Management tab, and select your organization.**

3 **From the View menu, select Services.**

4 **In the navigation frame, under Authentication, click Core.**

5 **Add the following to the** `Pluggable User Name Generator Class` **attribute:**
`com.iplanet.am.samples.authentication.spi.genuid. UserIDGeneratorSample`

6 **Click Save to save the changes.**

7 **Log out.**

▼ To Access an Authentication Module for an Organization

This module is the one which invokes the `UserIDGenerator` SPI implementation class. By default, only the Membership/Self-registration authentication module calls this SPI implementation.

1. **Make sure that you have registered and enabled the Membership authentication module, and that you have created a template for the organization.**

2. **Enter the following URL:**

   `http://`*host.domain:port/Service-Deploy-URI/UI/*`Login?module=Membership`

   If you choose to use an organization other than the default, be sure to specify that in the URL using the `org` parameter. Example: `org=`*orgName*

3. **Click New User.**

   You should be able to register any existing username or user ID.

   The `UserIDGeneratorSample` will be executed. You will be presented with the generated User IDs choice menu to choose any one username or user ID.

Compiling the UserIDGeneratorSample Program on Windows 2000

In the `<install-root>\samples\authentication\spi\genuid` directory, run the `make` command.

▼ To deploy the UserIDGeneratorSample Program

1. **Copy** `UserIDGeneratorSample.jar` **from** `JAR_DIR` **to** <*install-root*>*lib*

2. **In the Web Container from which this sample has to run, update the classpath with** `UserIDGeneratorSample.jar`**.**

3. **Restart Access Manager.**

   ```
   <install-root>\bin\amserver start
   ```

To Configure the UserIDGeneratorSample Program

Configuring the program on Windows 2000 is similar to configuring the program on Solaris. See "Configuring the Authentication Post Processing SPI" on page 56.

Implementing A Pure JAAS Module

A sample program demonstrates how to write pure a JAAS module to replay callbacks by authenticating using Access Manager Authentication Client API. It will authenticate a user by replaying the callbacks required by Access Manager the Authentication Module. You can modify this program to use other existing or customized Access Manager Authentication modules. This sample module can be plugged in into any standard JAAS framework using the JAAS API.

Note – For detailed information on JAAS, see the Sun Developer Documentation at the following URL:`http://java.sun.com/products/jaas/`. For detailed information on how to write a JAAS module, see the *JAAS LoginModule Developer's Guide* at the following URL:

```
http://java.sun.com/j2se/1.4.2/docs/guide/security/
                jaas/JAASLMDevGuide.html
```

Conventions Used in the Samples

TABLE 2–5 Default directories for Solaris Sparc/x86

Variable	Description	Location
Config_directory	Directory that contains configuration files	`/CONFIG_DIR = /etc/opt/SUNWam/config`
Product_Directory	Directory where Access Manager is installed.	`PRODUCT_DIR = <install_root>/SUNWam`

TABLE 2–6 Default directories for Linux

Variable	Description	Location
Config_Directory	Directory that contains configuration files	`CONFIG_DIR = /etc/opt/sun/identity/config`
Product_Directory	Directory where Access Manager is installed.	`PRODUCT_DIR = <install_root>/sun/identity`

TABLE 2–7 Default directories for Windows 2000

Variable	Description	Location
Config_Directory	Directory that contains configuration files	`CONFIG_DIR = <install_root>\lib`

TABLE 2–7 Default directories for Windows 2000 *(Continued)*

Variable	Description	Location
Product_Directory	Directory where Access Manager is installed.	

▼ To Run the Sample on Solaris Sparc x86 or Linux:

1. **In the Makefile, set the following variables:**

   BASE: Enter the path to the directory where Access manager is installed.

   JAVA_HOME: Enter the path to the directory where Java compiler is installed

   CONFIG: Enter the entry specified in the login configuration file. This entry will be used to do the user authentication

2. **Copy** `AMConfig.properties` **from Access Manager server installation machine location** `<CONFIG_DIR>` **to the client machine where the sample will be run.**

3. **On the client machine, be sure the following are in your classpath:**

   - `am_services.jar`
   - `jaas.jar`
   - `jss3.jar`
   - `AMConfig.properties`

     Include `jaas.jar` in your classpath if you are using a JDK version less than JDK1.4

4. **A sample configuration file** `purejaassample.config` **is provided for testing this sample.**

   The file contains only one entry named Sample. Sample is the name to be entered for `CONFIG` in the `Makefile`:

   ```
   Sample {
    PureJAASSampleLoginModule required ORG_NAME="dc=iplanet,dc=com"
              INDEX_NAME="LDAP" debug=true;
    };
   ```

   The entry specifies that the LoginModule to be used to do the user authentication is the `PureJAASSampleLoginModule` and that this `SampleLoginModule` must succeed in order for authentication to be considered successful. It passes options with `ORG_NAME` as the organization name and `INDEX_NAME` as the Access Manager authentication module to which this sample must authenticate.

   If you must use a different login configuration, modify the Makefile. For example, change the following:

   ```
   -Djava.security.auth.login.config=purejaassample.config
   ```

to this:

`-Djava.security.auth.login.config=`*your_jaas_config_file* *.config*

5 **To compile, run the** `gmake` **command.**

6 **To run the sample program run the** `gmake run` **command.**

▼ To Enable SSL

1 **In the sample client program, add this JVM property:**

```
-D "java.protocol.handler.pkgs=com.iplanet.services.comm"
```

2 **In the** `AMConfig.properties` **file, edit the following properties:**

com.iplanet.am.admin.cli.certdb.dir: `<PRODUCT_DIR>/servers/alias`

com.iplanet.am.admin.cli.certdb.prefix: `https-machine1.com-machine1-`

com.iplanet.am.server.protocol: `https`

com.iplanet.am.server.port: Enter the appropriate port on the server machine where `machine1` is the host name of the server

▼ To Run the Sample on Windows 2000

1 **In** `make.bat`**, set the following properties:**

BASE: Enter the path to the directory where Access manager is installed

JAVA_HOME: Enter the path to the directory where the Java compiler is installed.

CONFIG: Enter the entry which will be used for user authentication. This entry is specified in the login configuration file.

2 **Copy** `AMConfig.properties` **from Access Manager server installation machine location** `<CONFIG_DIR>` **to the client machine where this sample will be run.**

3 **On the client machine, make sure the following are in your classpath:**

- `am_services.jar`
- `jaas.jar`
- `jss3.jar`
- `AMConfig.properties`

  Include `jaas.jar` in your classpath if you are using JDK version less than JDK1.4.

4 **A sample configuration file** `purejaassample.config` **is provided for testing this sample.**

The file contains only one entry `named.Sample`. Sample is the name to be entered for `CONFIG` in the Makefile.

```
Sample {
 PureJAASSampleLoginModule required ORG_NAME="dc=iplanet,dc=com"
           INDEX_NAME="LDAP" debug=true;
 };
```

The entry specifies that the LoginModule to be used to do the user authentication is the `PureJAASSampleLoginModule. SampleLoginModule` is must succeed in order for authentication to be considered successful. It passes options with `ORG_NAME` as the organization name and `INDEX_NAME` as the Access Manager authentication module to which this sample has to authenticate.

If you must use a different login configuration, modify the Makefile. For example, change the following:

```
-Djava.security.auth.login.config=purejaassample.config
```

to this:

```
-Djava.security.auth.login.config=your_jaas_config_file.config
```

5 **To compile, run the** `make` **command.**

6 **To run the sample program, run the** `make run` **command.**

▼ To Enable SSL

1 **In the sample client program, add this JVM property:**

```
-D "java.protocol.handler.pkgs=com.iplanet.services.comm"
```

2 **Edit the following properties in the** `AMConfig.properties` **file:**

com.iplanet.am.admin.cli.certdb.dir:
`<install-dir>\SUN\IdentityServer6\Servers\alias`

com.iplanet.am.admin.cli.certdb.prefix:`https-machine1.red.iplanet.com-machine1-`

com.iplanet.am.server.protocol: `https`

com.iplanet.am.server.port: Enter the appropriate port on the server machine where `machine1` is the host name of the server

For the detailed information, see the Javadoc for Remote Client APIs. By default, Access Manager Javadoc is installed in the following directory:

AccessManager-base`/SUNWam/docs`

For the detailed information on how to plug the Login Module into the standard JAAS Context, see the *JAAS Reference Guide* at the following URL:

```
http://java.sun.com/j2se/1.5.0/docs/guide/security/jaas/JAASRefGuide.html
```

◆ ◆ ◆ CHAPTER 3

Using the Policy APIs

The Sun Java™ System Access Manager 7.1 Policy Service enables you to define, manage, and enforce policies that control access to protected resources. Administrators use the Policy Service to configure and manage conditions for applications, resources, and identities managed within the Access Manager deployment. For detailed information about what the Policy Service does and how it works, see *Chapter 4, "Authorization and the Policy Service," in Sun Java System Access Manager 7.1 Technical Overview*.

This chapter provides information about the Policy APIs and how to use them to enable your service to use Access Manager policies. The chapter includes the following topics:

About the Policy APIs

The Policy Java APIs enable you to do the following:

- Develop and add custom subjects, referrals, and conditions to Access Manager.
- Develop and run policy evaluation programs
- Programmatically construct policies and add them to the policy store.

This chapter describes Java Policy Service packages and classes, and provides instructions for using the Policy APIs and code samples. For a comprehensive listing of Policy Java methods and their usage, see the *Sun Java System Access Manager 7.1 Java API Reference*.

Access Manager also provides C APIs to enable external applications to connect to the Policy Service framework. For information about using the Policy C APIs, see *Chapter 3, "Policy Data Types and Functions," in Sun Java System Access Manager 7.1 C API Reference Chapter 3, "Policy Data Types and Functions," in Sun Java System Access Manager 7.1 C API Reference*.

Policy Java Packages

The following Java packages comprise the Policy APIs:

`com.sun.identity.policy`	Contains policy evaluation classes for policy administration and evaluation. Policy evaluation classes from this package require a direct connection to the policy data store. These classes should be used with caution, and only when classes from `com.sun.identity.policy.client` cannot handle your use case.
`com.sun.identity.policy.client`	Contains classes used by remote Java applications to evaluate policies and to get policy decisions.
`com.sun.identity.policy.interfaces`	Contains interfaces for writing custom Policy plug-ins for conditions, subjects, referrals and resources.

Policy Management Classes

Policy Management classes are used by system administrators to manage policies in Access Manager. The interfaces for this functionality are contained in the `com.sun.identity.policy` package and including the following:

- "PolicyManager" on page 68
- "Policy" on page 69

PolicyManager

`com.sun.identity.policy.PolicyManager` is the top-level administrator class for policy management. `com.sun.identity.policy.PolicyManager` provides methods that enable an administrator to create, modify, or delete realm policies. The `PolicyManager` can be obtained by passing a privileged user's session token or by passing a privileged user's session token with a realm name. Some of the more widely used methods of this class include the following:

`getPolicyNames`	Retrieves all named policies created for the realm for which the policy manager was instantiated. This method can also take a pattern (filter) as an argument.

`getPolicy`	Retrieves a policy when given the policy name.
`addPolicy`	Adds a policy to the specified realm. If a policy with the same name already exists, it will be overwritten.
`removePolicy`	Removes a policy from the specified realm.
`replacePolicy`	Replaces policy with a new policy.

Policy

`com.sun.identity.policy.Policy` represents a policy definition with all its intended parts (rules, subjects, referrals, conditions, and response providers). The policy object is saved in the data store if the `addPolicy` or `replacePolicy` methods from the `PolicyManager` class are invoked. This class contains methods for adding, removing, replacing or getting any of the parts of a policy definition.

Policy Evaluation Classes

Policy evaluation classes compute policy decisions which allow or deny access to a protected resource. Policy evaluation classes are contained `com.sun.identity.policy` package and include the following:

- PolicyEvaluator
- ProxyPolicyEvaluator
- ClientPolicyEvaluator
- PolicyEvent

PolicyEvaluator Class

`com.sun.identity.policy.PolicyEvaluator` can be integrated into Java applications to evaluate policy privileges and provide policy decisions. This class provides support for both boolean and non-boolean type policies. Create a `PolicyEvaluator` by calling the constructor with a service name. Public methods of this class include the following:

`isAllowed`	Evaluates the policy associated with the given resource and returns a boolean value indicating whether the policy evaluation resulted in an allow or deny.
`getPolicyDecision`	Evaluates policies and returns decisions. Returns a decision that gives a user permission to perform specified actions on a specified resource.
`getResourceResult`	Obtains the policy and decisions for a hierarchy of resources. Possible values for the scope of this method are `self`, `subtree`, and `strict-subtree`. Use the `self` value to get the policy decision for the specified resource only. Use the `subtree` value to include the policy

decisions for all resources defined in the policies which are sub-resources of the specified resource.

For example, the `PolicyEvaluator` class can be used to display the links for a list of resources to which an authenticated user has access. The `getResourceResult` method is used to get the list of resources. The `resourceName` parameter would be `http://`*host.domain:port* which returns all the resources to which the user has access on that server. These resources are returned as a `PolicyDecision` based on the user's defined policies. If the user is allowed to access resources on different servers, this method needs to be called for each server.

Note – Not all resources that have policy decisions are accessible to the user. Access depends on `ActionDecision(s)` contained in policy decisions.

ProxyPolicyEvaluator Class

`com.sun.identity.policy.ProxyPolicyEvaluator` allows a privileged user (top-level administrator, organization administrator, policy administrator, or organization policy administrator) to get policy privileges and evaluate policy decisions for any user in their respective scope of administration.
`com.sun.identity.policy.ProxyPolicyEvaluatorFactory` is the singleton class used to get `ProxyPolicyEvaluator` instances.

EXAMPLE 3–1 Public Methods For ProxyPolicyEvaluator

```
/**
 * Evaluates a simple privilege of boolean type. The privilege
 * indicates if the user identified by the principalName
 * can perform specified action on the specified resource.
 *
 * @param principalName principal name for whom to
 * compute the privilege.
 * @param resourceName name of the resource
 * for which to compute policy result.
 * @param actionName name of the action the user is trying to
 * perform on the resource
 * @param env run time environment parameters
 *
 * @return the result of the evaluation as a boolean value
 *
 * @throws PolicyException exception form policy framework
 * @throws SSOException if sso token is invalid
 *
 */
public boolean isAllowed(String principalName, String resourceName,
    String actionName, Map env) throws PolicyException, SSOException;
```

EXAMPLE 3–1 Public Methods For ProxyPolicyEvaluator *(Continued)*

```
/**
 * Gets policy decision for the user identified by the
 * principalName for the given resource
 *
 * @param principalName principal name for whom to compute the
 * policy decision
 * @param resourceName name of the resource for which to
 * compute policy decision
 * @param env run time environment parameters
 *
 * @return the policy decision for the principal for the given
 * resource
 * @throws PolicyException exception form policy framework
 * @throws SSOException if sso token is invalid
 *
 */
public PolicyDecision getPolicyDecision(String principalName,
    String resourceName, Map env)
    throws PolicyException, SSOException;

/**
 * Gets protected resources for a user identified by the
 * principalName. Conditions defined in the policies
 * are ignored while computing protected resources.
 * Only resources that are  subresources of the  given
 * rootResource or equal to the  given rootResource would
 * be returned.
 * If all policies applicable to a resource are
 * only referral policies, no ProtectedResource would be
 * returned for such a resource.
 * @param principalName principal name for whom
 * to compute the privilege.
 * @param rootResource  only resources that are subresources
 * of the given rootResource or equal to the  given
 * rootResource would be returned. If
 * <code>PolicyEvaluator.ALL_RESOURCES</code>
 * is passed as rootResource, resources under
 * all root  resources of the service
 * type are considered while computing protected
 * resources.
 *
 * @return set of protected resources. The set contains
 * ProtectedResource objects.
 *
```

EXAMPLE 3–1 Public Methods For ProxyPolicyEvaluator *(Continued)*

```
     * @throws PolicyException exception form policy framework
     * @throws SSOException if sso token is invalid
     * @see ProtectedResource
     *
     */
    public Set getProtectedResourcesIgnoreConditions(String principalName,
        String rootResource)  throws PolicyException, SSOException
```

Client PolicyEvaluator Class

`com.sun.identity.policy.client.PolicyEvaluator` evaluates policies and provides policy decisions for remote applications. This does not require direct access to a policy stores such as Directory Server (for example, if there is a firewall).
`com.sun.identity.policy.client.PolicyEvaluator` get policy decision from Access Manager using `XML` over `HTTP(s)`. It stores a cache of policy decisions for faster responses and maintains the cache in sync with the Policy Service on the instance of Access Manager using the notification and polling mechanism.

PolicyEvent Class

`com.sun.identity.policy.PolicyEvent` represents a policy event that could potentially change the current access status. For example, a policy event is created and passed to the registered policy listeners whenever there is a change in a policy rule. This class works with the `PolicyListener` class in the `com.sun.identity.policy.interface` package.

Policy Plug-In APIs

The Policy plug-in classes are contained in the `com.sun.identity.policy.interfaces` package. The following classes are used by service developers and policy administrators who need to provide additional policy features as well as support for legacy policies.

`ResourceName`	Provides methods to determine the hierarchy of the resource names for a determined service type. For example, these methods can check to see if two resources names are the same or if one is a sub-resource of the other.
`Subject`	Defines methods that can determine if an authenticated user (possessing an `SSOToken`) is a member of the given subject.
`Referral`	Defines methods used to delegate the policy definition or evaluation of a selected resource (and its sub-resources) to another realm or policy server.

`Condition`	Provides methods used to constrain a policy to , for example, time-of-day or IP address. This interface allows the pluggable implementation of the conditions.
`PolicyListener`	Defines an interface for registering policy events when a policy is added, removed or changed. `PolicyListener` is used by the Policy Service to send notifications and by listeners to review policy change events.

Using the Policy Code Samples

Access Manager provides Policy code samples to perform the following tasks:

- Add a new service which has a policy schema to Access Manager
- Develop and add custom developed subjects, referrals, conditions and response providers to Access Manager
- Develop and run Policy evaluation programs
- Construct policies programmatically and add them to the policy store
- Create policies using amadmin command

All the files you need to run the policy code samples are located in the following directories:

Solaris Platform	*AccessManager-base*`\samples\policy`
Linux and HP-UX Platforms	*AccessManager-base*`\identity\samples\policy`
Windows Platform	*AccessManager-base*`/identity/samples/policy`

Use Cases Illustrated by Policy Code Samples

Each of the following sections describes a sequence of steps you must take when using various means to run a policy evaluation program or to create policies. Each step in a sequence is linked to detailed instructions further down in this chapter.

▼ To Run a Policy Evaluation Program for the URL Policy Agent Service

Use this sequence to runs a policy evaluation program for the `iPlanetAMWebAgentService` service.

1 **Compile the Policy code samples.**

See Compiling the Policy Code Samples.

2 **Develop and run a Policy evaluation program.**

See "Developing and Running a Policy Evaluation Program" on page 87.

▼ To Run a Policy Evaluation Program for the URL Policy Agent Service and More

This sequence runs the evaluation program for `iPlanetAMWebAgentService` and the sample subject, condition, ResponseProvider, and referral implementations.

1 **Compile the Policy code samples.**

See "Compiling the Policy Code Samples" on page 76.

2 **Develop custom subjects, conditions, and referrals.**

See "Developing Custom Subjects, Conditions, Referrals, and Response Providers" on page 80.

3 **Develop and run a Policy evaluation program.**

See "Developing and Running a Policy Evaluation Program" on page 87.

▼ To Run a Policy Evaluation Program for the Sample Service

This sequence runs the evaluation program for the `SampleWebService`.

1 **Compile the Policy code samples.**

See "Compiling the Policy Code Samples" on page 76.

2 **Add a Policy-enabled service to Access Manager.**

See "Adding a Policy-Enabled Service to Access Manager" on page 76.

3 **Create policies for the new service.**

See "Creating Policies for a New Service" on page 86.

4 **Develop and run a Policy evaluation program.**

"Developing and Running a Policy Evaluation Program" on page 87.

▼ To Run a Policy Evaluation Program for the Sample Service and More

This sequence runs the evaluation program for`SampleWebService` and the sample subject, condition, Response Provider, and referral implementations.

1 **Compile the Policy code samples.**

See "Compiling the Policy Code Samples" on page 76.

2 **Add a Policy-enabled service to Access Manager.**

See "Adding a Policy-Enabled Service to Access Manager" on page 76.

3 **Develop custom subjects, conditions, and referrals.**

See "Developing Custom Subjects, Conditions, Referrals, and Response Providers" on page 80.

4 **Create policies for the new service.**

See "Creating Policies for a New Service" on page 86.

5 **Develop and run a Policy evaluation program.**

See "Developing and Running a Policy Evaluation Program" on page 87.

To Use amadmin to Create Policies for the URL Policy Agent Service

Use amadmin to create policies for the service. See "Creating Policies" in *Sun Java System Access Manager 7.1 Administration Guide* for detailed instructions.

▼ To Use amadmin to Create Policies for the Sample Service

This sequences creates policies for `SampleWebService`.

1 **Compile the Policy code samples.**

See "Compiling the Policy Code Samples" on page 76.

2 **Develop and run a Policy evaluation program.**

See "Developing and Running a Policy Evaluation Program" on page 87.

▼ To Programmatically Construct Policies

This sequence constructs policies and adds them to the policy store.

1. **Compile the Policy code samples.**
   See "Compiling the Policy Code Samples" on page 76.
2. **Programmatically construct policies.**
   See "Programmatically Constructing Policies" on page 88.

Compiling the Policy Code Samples

Samples can be run both on Solaris, Linux, HP-UX, and Windows platforms. In the sample files, root suffix `DNs` are specified as `dc=example,dc=com`. Substitute the root suffix with the actual root suffix of your Access Manager installation.

▼ To Compile the Policy Code Samples

1. **Set the following variables in the `Makefile` (or `make.bat` in Windows):**

   `BASE` Set this to refer the directory where Access Manager is installed.

   `JAVA_HOME` Set this variable to your installation of JDK. The JDK version should be higher than JDK 1.4
2. **To compile the sample program, run the `gmake all` command (or `make.bat` in Windows).**
3. **In the sample files, replace the root suffix `DNs` with values appropriate for your environment.**

Adding a Policy-Enabled Service to Access Manager

You can load into Access Manager a service that already contains policy schema. Access Manager provides a sample XML file for a new service that contains policy schema. You can modify *AccessManager-base*`/SUNWam/samples/policy/SampleWebService.xml` to fit your needs, and then add your service to Access Manager.

The Policy element contains `AttributeSchema` elements to define applicable actions and values for actions. While defining policies, you can define access rules for those actions.

Examples include `canForwardEmailAddress` and `canChangeSalaryInformation` . The actions specified by these attributes can be associated with a resource if the `IsResourceNameAllowed` element is specified in the attribute definition. For example, in the web agent XML service file, `amWebAgent.xml` , `GET` and `POST` are defined as policy attributes with an associated URL resource as `IsResourceNameAllowed` is specified.

EXAMPLE 3–2 SampleWebService.xml

```
<!DOCTYPE ServicesConfiguration
    PUBLIC "=//iPlanet//Service Management Services (SMS) 1.0 DTD//EN"
    "jar://com/sun/identity/sm/sms.dtd">

<ServicesConfiguration>
    <Service name="SampleWebService" version="5.0">
        <Schema
            serviceHierarchy="/DSAMEConfig/SampleWebService"
            i18nFileName="SampleWebService"
            i18nKey="SampleWebService">*
        <Global>
        <AttributeSchema name="serviceObjectClasses"  type="list" syntax="string"
                                i18nKey="SampleWebService"/>
            </Global>
        <Policy>
        <AttributeSchema name="GET"
                    type="single"
                    syntax="boolean"
                    uitype="radio"
                    i18nKey="get">
                    <IsResourceNameAllowed/>
                    <BooleanValues>
                        <BooleanTrueValue i18nKey="allow">allow</BooleanTrueValue>
                        <BooleanFalseValue i18nKey="deny">deny</BooleanFalseValue>
                    </BooleanValues>
        </AttributeSchema>

        <AttributeSchema name="POST"
            type="single"
                    syntax="boolean"
                    uitype="radio"
            i18nKey="post">
            <IsResourceNameAllowed/>
                    <BooleanValues>
                        <BooleanTrueValue i18nKey="allow">allow</BooleanTrueValue>
                        <BooleanFalseValue i18nKey="deny">deny</BooleanFalseValue>
                    </BooleanValues>
        </AttributeSchema>
```

EXAMPLE 3–2 SampleWebService.xml *(Continued)*

```
        <AttributeSchema name="PUT"
            type="single"
                    syntax="boolean"
                    uitype="radio"
            i18nKey="put">
            <IsResourceNameAllowed/>
                    <BooleanValues>
                        <BooleanTrueValue i18nKey="allow">allow</BooleanTrueValue>
                        <BooleanFalseValue i18nKey="deny">deny</BooleanFalseValue>
                    </BooleanValues>
        </AttributeSchema>

        <AttributeSchema name="DELETE"
            type="single"
                    syntax="boolean"
                    uitype="radio"
            i18nKey="delete">
            <IsResourceNameAllowed/>
                    <BooleanValues>
                        <BooleanTrueValue i18nKey="allow">allow</BooleanTrueValue>
                        <BooleanFalseValue i18nKey="deny">deny</BooleanFalseValue>
                    </BooleanValues>
        </AttributeSchema>

        </Policy>
        </Schema>
    </Service>
</ServicesConfiguration>
```

▼ To Add a New Policy-Enabled Service to Access Manager

1 Run the amadmin command to load the policy-enabled service.

AccessManager-base`/bin/amadmin`
`--runasdn "uid=amAdmin,ou=People,`*default_org*`,`*root_suffix*`"`
`--password` *password*
`--schema` *AccessManager-base*`/samples/policy/`*SampleWebService.xml*

2 Copy the properties file to the `locale` directory of the Access Manager installation.

`cp` *SampleWebService.properties* *AccessManager-base*`/locale`

3. **Create a service XML file that conforms to** *AccessManager-base*`/dtd/sms.dtd`**, and contains the** `<Policy>` **element. See example below.**

4. **Create and copy locale properties file to** *AccessManager-base*`/locale`**.**

5. **Use** `amadmin` **to load the service into Access Manager.**

   Once the new service is added, you can define rules for the new service in policy definitions.

Example 3–3 XML for Policy-Enabled Service

`/etc/opt/SUNWam/config/xml/amWebAgent.xml` (Solaris)
`/etc/opt/sun/identity/config/xml/amWebAgent.xml`(Linux and HP-UX)

AccessManager-base`\AccessManager\identity\config\xml\amWebAgent.xml`

(Windows)

```
<!DOCTYPE ServicesConfiguration
    PUBLIC "=//iPlanet//Service Management Services (SMS) 1.0 DTD//EN"
    "jar://com/sun/identity/sm/sms.dtd">

<ServicesConfiguration>
    <Service name="iPlanetAMWebAgentService" version="1.0">
    <Schema
            i18nFileName="amWebAgent"
            i18nKey="iplanet-am-web-agent-service-description">
    <Global>
        <AttributeSchema name="serviceObjectClasses"
                    type="list"
                    syntax="string"
    i18nKey="">
                    <DefaultValues>
            <Value>iplanet-am-web-agent-service</Value>
    </DefaultValues>
        </AttributeSchema>
            </Global>

            <Policy>
        <AttributeSchema name="GET"
                    type="single"
    syntax="boolean"
                    uitype="radio"
                    i18nKey="GET">
    <IsResourceNameAllowed/>
                    <BooleanValues>
            <BooleanTrueValue i18nKey="allow">allow</BooleanTrueValue>
```

```
            <BooleanFalseValue i18nKey="deny">deny</BooleanFalseValue>
                    </BooleanValues>
        </AttributeSchema>
        <AttributeSchema name="POST"
                    type="single"
    syntax="boolean"
                    uitype="radio"
                    i18nKey="POST">
    <IsResourceNameAllowed/>
                    <BooleanValues>
                   <BooleanTrueValue i18nKey="allow">allow</BooleanTrueValue>
            <BooleanFalseValue i18nKey="deny">deny</BooleanFalseValue>
                    </BooleanValues>
        </AttributeSchema>
            </Policy>
    </Schema>
    </Service>
</ServicesConfiguration>
```

Developing Custom Subjects, Conditions, Referrals, and Response Providers

Access Manager provides subject, condition, referral, and response provider interfaces that enable you to develop your own custom subjects, conditions, referrals, and response providers. A sample implementation is provided for the following four interfaces.

`SampleSubject.java`	Implements the `Subject` interface. This subject applies to all the authenticated users who have valid `SSOTokens`.
`SampleCondition.java`	Implements the `Condition` interface. This condition makes the policy applicable to those users whose user name length is greater than or equal to the length specified in the condition.
`SampleReferral.java`	Implements the `Referral` interface. `SampleReferral.java` gets the referral policy decision from a text file `SampleReferral.properties` located in the `/samples` directory.
`SampleResponseProvider.java`	Implements the `ResponseProvider` interface. `SampleResponseProvider.java` takes as input the attribute for which values are retrieved from the Access Manager and sent back in the Policy Decision. If the attribute does not exist in the user profile, no value is sent back in the response. `SampleResponseProvider.java`

relies on the underlying Identity Repository service to retrieve the attribute values for the Subject(s) defined in the policy.

You must add the subject, condition, response provider, referral implementations to `iPlanetAMPolicyService` and `iPlanetAMPolicyConfigService` in order to make them available for policy definitions. These services are loaded into Access Manager during installation. To add the sample implementations to the Policy framework, modify the `iPlanetAMPolicy` service and `iPlanetAMPolicyConfig` service. The service XML files are located in the following directory:

AccessManager-base`/SUNWam/samples/policy`

The following is the text of the `amPolicy_mod.xml` file for the `iPlanetAMPolicy` service .

EXAMPLE 3–4 Text of the Default `amPolicy_mod.xml` File

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Copyright (c) 2005 Sun Microsystems, Inc. All rights reserved
    Use is subject to license terms.
-->


<!DOCTYPE ServicesConfiguration
    PUBLIC "=//iPlanet//Service Management Services (SMS) 1.0 DTD//EN"
    "jar://com/sun/identity/sm/sms.dtd">

<ServicesConfiguration>
    <Service name="iPlanetAMPolicyService" version="1.0">
        <PluginSchema className="SampleSubject"
                      i18nFileName="amPolicy"
                      i18nKey="iplanet-subject-SampleSubject-name"
                      interfaceName="Subject"
                      name="SampleSubject" >
        </PluginSchema>

        <PluginSchema className="SampleCondition"
                      i18nFileName="amPolicy"
                      i18nKey="iplanet-samplecondition-condition-name"
                      interfaceName="Condition"
                      name="SampleCondition" >
        </PluginSchema>

        <PluginSchema className="SampleReferral"
                      i18nFileName="amPolicy"
```

EXAMPLE 3–4 Text of the Default `amPolicy_mod.xml` File *(Continued)*

```
                    i18nKey="iplanet-sample-referral"
                    interfaceName="Referral"
                    name="SampleReferral" >
        </PluginSchema>
        <PluginSchema className="SampleResponseProvider"
                    i18nFileName="amPolicy"
                    i18nKey="iplanet-sample-responseprovider"
                    interfaceName="ResponseProvider"
                    name="SampleResponseProvider" >
        </PluginSchema>
    </Service>
</ServicesConfiguration>
```

The following is the text of the `amPolicyConfig_mod.xml` file for the `iPlanetAMPolicyConfig` service .

EXAMPLE 3–5 Text of the Default `amPolicyConfig_mod.xml` File

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Copyright (c) 2005 Sun Microsystems, Inc. All rights reserved
    Use is subject to license terms.
-->



<!DOCTYPE Requests
    PUBLIC "-//iPlanet//Sun Java System Access Manager 2005Q4 Admin CLI DTD//EN"
 "jar://com/iplanet/am/admin/cli/amAdmin.dtd"
>

<Requests>

    <SchemaRequests serviceName="iPlanetAMPolicyConfigService"
        SchemaType="Organization"
        i18nKey="a163">
        <AddChoiceValues>
            <AttributeValuePair>
            <Attribute name="sun-am-policy-selected-responseproviders"/>
                <Value>SampleResponseProvider</Value>
            </AttributeValuePair>
        </AddChoiceValues>
    </SchemaRequests>
```

EXAMPLE 3–5 Text of the Default `amPolicyConfig_mod.xml` File *(Continued)*

```
<SchemaRequests serviceName="iPlanetAMPolicyConfigService"
    SchemaType="Organization"
    i18nKey="">
    <AddDefaultValues>
        <AttributeValuePair>
        <Attribute name="sun-am-policy-selected-responseproviders"/>
            <Value>SampleResponseProvider</Value>
        </AttributeValuePair>
    </AddDefaultValues>
</SchemaRequests>

<SchemaRequests serviceName="iPlanetAMPolicyConfigService"
    SchemaType="Organization"
    i18nKey="a160">
    <AddChoiceValues>
        <AttributeValuePair>
        <Attribute name="iplanet-am-policy-selected-subjects"/>
            <Value>SampleSubject</Value>
        </AttributeValuePair>
    </AddChoiceValues>
</SchemaRequests>

<SchemaRequests serviceName="iPlanetAMPolicyConfigService"
    SchemaType="Organization"
    i18nKey="">
    <AddDefaultValues>
        <AttributeValuePair>
        <Attribute name="iplanet-am-policy-selected-subjects"/>
            <Value>SampleSubject</Value>
        </AttributeValuePair>
    </AddDefaultValues>
</SchemaRequests>

<SchemaRequests serviceName="iPlanetAMPolicyConfigService"
    SchemaType="Organization"
    i18nKey="a161">
    <AddChoiceValues>
        <AttributeValuePair>
        <Attribute name="iplanet-am-policy-selected-conditions"/>
            <Value>SampleCondition</Value>
        </AttributeValuePair>
    </AddChoiceValues>
</SchemaRequests>

<SchemaRequests serviceName="iPlanetAMPolicyConfigService"
```

EXAMPLE 3–5 Text of the Default `amPolicyConfig_mod.xml` File *(Continued)*

```
        SchemaType="Organization"
        i18nKey="">
        <AddDefaultValues>
            <AttributeValuePair>
            <Attribute name="iplanet-am-policy-selected-conditions"/>
                <Value>SampleCondition</Value>
            </AttributeValuePair>
        </AddDefaultValues>
    </SchemaRequests>


    <SchemaRequests serviceName="iPlanetAMPolicyConfigService"
        SchemaType="Organization"
        i18nKey="a162">
        <AddChoiceValues>
            <AttributeValuePair>
            <Attribute name="iplanet-am-policy-selected-referrals"/>
                <Value>SampleReferral</Value>
            </AttributeValuePair>
        </AddChoiceValues>
    </SchemaRequests>

    <SchemaRequests serviceName="iPlanetAMPolicyConfigService"
        SchemaType="Organization"
        i18nKey="">
        <AddDefaultValues>
            <AttributeValuePair>
            <Attribute name="iplanet-am-policy-selected-referrals"/>
                <Value>SampleReferral</Value>
            </AttributeValuePair>
        </AddDefaultValues>
    </SchemaRequests>

</Requests>
```

▼ To Add a Sample Implementation to the Policy Framework

1 **Use `dscfg` to back up `iPlanetAMPolicy` and `iPlanetAMPolicyConfig` services.**

```
# cd DirectoryServer-base/ds6/bin
# ./dscfg export
-s "ou=iPlanetAMPolicyService,ou=services,root_suffix" output_file
```

```
# ./dscfg export
-s "ou=iPlanetAMPolicyConfigService,ou=services,root_suffix" output_file
```

2 Set the environment variable `LD_LIBRARY_PATH`.

On Solaris, add `/usr/lib/mps/secv1` to `LD_LIBRARY_PATH`.

On Linux, add `/opt/sun/private/lib` to `LD_LIBRARY_PATH`.

On HP-UX, add `/opt/sun/private/lib` to `SHLIB_PATH`.

3 Run the following commands:

```
# cd AccessManager-base/samples/policy
      AccessManager-base/bin/amadmin
--runasdn "uid=amAdmin,ou=People,default_org,root_suffix
--password password
--schema amPolicy_mod.xml
      AccessManager-base/bin/amadmin
--runasdn "uid=amAdmin,ou=People,default_org,root_suffix
--password password
--data amPolicyConfig_mod.xml
```

4 Change the properties files of the `iPlanetAMPolicy` and `iPlanetAMPolicyConfig` services to add messages related to the new implementations.

```
# cd AccessManager-base/locale
      cp amPolicy.properties amPolicy.properties.orig
      cp amPolicy_en.properties amPolicy_en.properties.orig
      cp amPolicyConfig.properties amPolicyConfig.properties.orig
      cp amPolicyConfig_en.properties amPolicyConfig_en.properties.orig
      cat <BASE_DIR>/samples/policy/amPolicy.properties  >>
          <BASE_DIR>/locale/amPolicy.properties
      cat <BASE_DIR>/samples/policy/amPolicy_en.properties >>
          <BASE_DIR>/locale/amPolicy_en.properties
      cat <BASE_DIR>/samples/policy/amPolicyConfig.properties  >>
          <BASE_DIR>/locale/amPolicyConfig.properties
      cat <BASE_DIR>/samples/policy/amPolicyConfig_en.properties >>
          <BASE_DIR>/locale/amPolicyConfig_en.properties
```

5 Deploy the sample plug-ins.

Copy `SampleSubject.class` , `SampleCondition.class`, `SampleResponseProvider.class`, `SampleReferral.class` from the `/samples/policy` directory to *AccessManager-base*`/lib` .

6 Restart the Access Manager server.

The sample subject, condition, response provider, and referral implementations are now available for policy definitions through the administration console or `amadmin` tool.

Creating Policies for a New Service

Access Manager policies are managed through the Administration console or through the `amadmin` command. However, policies cannot be modified using `amadmin` command. You must delete the policy, and then add the modified policy using `amadmin`. To add policies using amadmin, the Policy XML file must be developed following *AccessManager-base*`/dtd/policy.dtd`. Once the Policy XML file is developed, you can load the Policy XML file.

Two sample Policy XML files exist in the Policy `/samples` directory. The sample Policy XML files define policies for the`SampleWebService` service. `SamplePolicy.xml` defines a normal policy for `SampleWebService` with a `SampleSubject`, a `SampleResponseProvider`, and a `SampleCondition`. `SamplereferralPolicy.xml` defines a referral policy for `SampleWebService` with a `SampleReferral`.

▼ To Load a Policy XML File

Before You Begin You must compile the Policy code samples and develop custom subjects, conditions, response providers, and referrals before you can load policies present in the Policy XML files. See "Compiling the Policy Code Samples" on page 76 and "Developing Custom Subjects, Conditions, Referrals, and Response Providers" on page 80 for detailed instructions.

1 Run the following command:

AccessManager-base`/bin/amadmin`
`--runasdn "uid=amAdmin,ou=People,`*<default_org>*`,`*root_suffix*`>"`
`--password <`*password*`>`
`--data <`*policy.xml*`>`

2 Run the following command:

AccessManager-base`/bin/amadmin`
`--runasdn "uid=amAdmin,ou=People,`*default_org*`,`*root_suffix*`"`
`--password` *password*
`--data` *AccessManager-base*`/samples/policy/SamplePolicy.xml`
AccessManager-base`/bin/amadmin`
`--runasdn "uid=amAdmin,ou=People,`*default_org*`,`*root_suffix*`"`
`--password` *password*
`--data` *AccessManager-base*`/samples/policy/`
`SampleReferralPolicy.xml`

You can verify the newly added policies in Administration Console.

Developing and Running a Policy Evaluation Program

Access Manager provides a Policy Evaluation API. This API has one Java class, `PolicyEvaluator`. The package for this class is `com.sun.identity.policy.PolicyEvaluator`. Access Manager provides a sample policy evaluator program, `PolicyEvaluation.java`. You can use this program to run policy evaluations for different services. The policy evaluation is always based on a service such as `iPlanetAMWebAgentService` or `SampleWebService`. The sample policy evaluation program uses the `PolicyEvaluation.properties` file. Specify the input for the evaluation program in this file. Examples are service name, action names, condition environment parameters, user name, and user password.

▼ To Set Policy Evaluation Properties

1. **Set the value of `pe.servicename` to the service name.**

   Examples: `iPlanetAMWebAgentService` or `SampleWebService`.

2. **Set the `pe.resoucename` to the name of the resource that you want to evaluate the policy against.**

3. **Specify the action names in the `pe.actionnames`.**

   Separate the action names with a colon (:) If you want to get all the action values, leave the `pe.actionnames`blank.

4. **Set other required properties such as `pe.username` and `pe.password`.**

5. **(Optional) Set the following properties `pe.authlevel`, `pe.authscheme`, `pe.requestip`, `pe.dnsname`, `pe.time` if you use the corresponding conditions in your policy definitions.**

   If you don't want to set these environment parameters, just leave their values as blank.

`pe.authlevel`	Used to evaluate `AuthLevel` Condition. `pe.authlevel` takes a positive integer.
`pe.authscheme`	Used to evaluate `AuthScheme` Condition. `pe.authscheme`takes a set of colon— separated `AuthScheme` names.
`pe.requestip`	Used to evaluate the `IP` Condition. `pe.requestip` takes an IP address string.
`pe.dnsname`	Used to evaluate the `IP` Condition. `pe.dnsname` takes a set of colon—separated DNS names.
`property pe.time`	Used to evaluate the Simple Time Condition. `property pe.time` specifies the request time in milliseconds. If its value is set to the current time, then it takes the current time in milliseconds.

▼ To Run a Policy Evaluation Program

Before You Begin You must set up policies before running a policy evaluation program.

1. **Set the environment variable** `LD_LIBRARY_PATH`**.**

   On Solaris, add `/usr/lib/mps/secv1` to `LD_LIBRARY_PATH` .

   On Linux, add `/opt/sun/private/lib` to `LD_LIBRARY_PATH`.

   On HP-UX, add `/opt/sun/private/lib` to the environment variable `SHLIB_PATH`.

2. **Run the** `gmake run` **command (On Windows,** `make.bat run`**).**

Programmatically Constructing Policies

Access Manager provides Policy Management APIs that enable you to programmatically create, add, update and remove policies. The sample program `PolicyCreator.java` demonstrates how to programmatically construct policies and add them to policy store. The program creates one normal policy named `policy1` and one referral policy named `refpolicy1` and adds both policies to the policy store. The normal policy has one subject of each subject type, one condition of each condition type, and one response provider of each response provider type that comes with Access Manager at installation.

EXAMPLE 3–6 Sample Program `PolicyCreator.java`

```
/**
 * $Id: PolicyCreator.java,v 1.5 2005/06/24 16:53:50 vs125812 Exp $
 * Copyright © 2005 Sun Microsystems, Inc.  All rights reserved.
 *


import com.sun.identity.policy.PolicyManager;
import com.sun.identity.policy.ReferralTypeManager;
import com.sun.identity.policy.SubjectTypeManager;
import com.sun.identity.policy.ConditionTypeManager;
import com.sun.identity.policy.Policy;
import com.sun.identity.policy.Rule;
import com.sun.identity.policy.interfaces.Referral;
import com.sun.identity.policy.interfaces.Subject;
import com.sun.identity.policy.interfaces.Condition;
import com.sun.identity.policy.PolicyException;

import com.iplanet.sso.SSOToken;
import com.iplanet.sso.SSOException;
```

EXAMPLE 3–6 Sample Program `PolicyCreator.java` *(Continued)*

```
import java.util.Set;
import java.util.HashSet;
import java.util.Map;
import java.util.HashMap;

public class PolicyCreator {

    public static final String DNS_NAME="DnsName";
    public static final String DNS_VALUE="*.red.iplanet.com";
    public static final String START_TIME="StartTime";
    public static final String START_TIME_VALUE="08:00";
    public static final String END_TIME="EndTime";
    public static final String END_TIME_VALUE="21:00";
    public static final String AUTH_LEVEL="AuthLevel";
    public static final String AUTH_LEVEL_VALUE="0";
    public static final String AUTH_SCHEME="AuthScheme";
    public static final String AUTH_SCHEME_VALUE="LDAP";


    private String orgDN;
    private SSOToken ssoToken;
    private PolicyManager pm;

    private PolicyCreator() throws PolicyException, SSOException {
        BaseUtils.loadProperties();
        orgDN = BaseUtils.getProperty("pe.realmname");
        System.out.println("realmDN = " + orgDN);
        ssoToken = BaseUtils.getToken();
        pm = new PolicyManager(ssoToken, orgDN);
    }

    public static void main(String[] args) {
        try {
            PolicyCreator pc = new PolicyCreator();
            pc.addReferralPolicy();
            pc.addNormalPolicy();
            System.exit(0);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    private void addNormalPolicy() throws PolicyException, SSOException {
        System.out.println("Creating normal policy in realm:" + orgDN);
        PolicyManager pm = new PolicyManager(ssoToken, orgDN);
```

EXAMPLE 3–6 Sample Program `PolicyCreator.java` *(Continued)*

```
SubjectTypeManager stm = pm.getSubjectTypeManager();
ConditionTypeManager ctm = pm.getConditionTypeManager();

Policy policy = new Policy("policy1", "policy1 description");
Map actions = new HashMap(1);
Set values = new HashSet(1);
values.add("allow");
actions.put("GET", values);
String resourceName = "http://myhost.com:80/hello.html";
Rule rule = new Rule("rule1", "iPlanetAMWebAgentService",
        resourceName, actions);
policy.addRule(rule);

Subject subject = stm.getSubject("Organization");
Set subjectValues = new HashSet(1);
subjectValues.add(orgDN);
subject.setValues(subjectValues);
policy.addSubject("organization", subject);

subject = stm.getSubject("LDAPUsers");
subjectValues = new HashSet(1);
String userDN = "uid=user1,ou=people" + "," + orgDN;
subjectValues.add(userDN);
subject.setValues(subjectValues);
policy.addSubject("ldapusers", subject);

subject = stm.getSubject("LDAPGroups");
subjectValues = new HashSet(1);
String groupDN = "cn=group1,ou=groups" + "," + orgDN;
subjectValues.add(groupDN);
subject.setValues(subjectValues);
policy.addSubject("ldapgroups", subject);

subject = stm.getSubject("LDAPRoles");
subjectValues = new HashSet(1);
String roleDN = "cn=role1" + "," + orgDN;
subjectValues.add(roleDN);
subject.setValues(subjectValues);
policy.addSubject("ldaproles", subject);

subject = stm.getSubject("IdentityServerRoles");
subjectValues = new HashSet(1);
roleDN = "cn=role1" + "," + orgDN;
subjectValues.add(roleDN);
subject.setValues(subjectValues);
```

EXAMPLE 3–6 Sample Program `PolicyCreator.java` *(Continued)*

```
        policy.addSubject("is-roles", subject);

        Condition condition = ctm.getCondition("IPCondition");
        Map conditionProperties = new HashMap(1);
        Set propertyValues = new HashSet(1);
        propertyValues.add(DNS_VALUE);
        conditionProperties.put(DNS_NAME, propertyValues);
        condition.setProperties(conditionProperties);
        policy.addCondition("ip_condition", condition);

        condition = ctm.getCondition("SimpleTimeCondition");
        conditionProperties = new HashMap(1);
        propertyValues = new HashSet(1);
        propertyValues.add(START_TIME_VALUE);
        conditionProperties.put(START_TIME, propertyValues);
        propertyValues = new HashSet(1);
        propertyValues.add(END_TIME_VALUE);
        conditionProperties.put(END_TIME, propertyValues);
        condition.setProperties(conditionProperties);
        policy.addCondition("time_condition", condition);

        condition = ctm.getCondition("AuthLevelCondition");
        conditionProperties = new HashMap(1);
        propertyValues = new HashSet(1);
        propertyValues.add(AUTH_LEVEL_VALUE);
        conditionProperties.put(AUTH_LEVEL, propertyValues);
        condition.setProperties(conditionProperties);
        policy.addCondition("auth_level_condition", condition);


        condition = ctm.getCondition("AuthSchemeCondition");
        conditionProperties = new HashMap(1);
        propertyValues = new HashSet(1);
        propertyValues.add(AUTH_SCHEME_VALUE);
        conditionProperties.put(AUTH_SCHEME, propertyValues);
        condition.setProperties(conditionProperties);
        policy.addCondition("auth_scheme_condition", condition);


        pm.addPolicy(policy);

        System.out.println("Created  normal policy");
    }

    private void addReferralPolicy()
```

EXAMPLE 3–6 Sample Program `PolicyCreator.java` *(Continued)*

```
            throws PolicyException, SSOException {
        System.out.println("Creating referral policy for realm1");
        ReferralTypeManager rtm = pm.getReferralTypeManager();
        String subOrgDN = "o=realm1" +  ",ou=services," + orgDN;
        Policy policy = new Policy("refpolicy1", "ref to realm1",
                true);
        Map actions = new HashMap(1);
        Rule rule = new Rule("rule1", "iPlanetAMWebAgentService",
                "http://myhost.com:80/realm1", actions);
        policy.addRule(rule);
        Referral referral = rtm.getReferral("SubOrgReferral");
        Set referralValues = new HashSet(1);
        referralValues.add(subOrgDN);
        referral.setValues(referralValues);
        policy.addReferral("ref to realm1" , referral);
        pm.addPolicy(policy);
        System.out.println("Created referral policy for realm1");
    }

}
```

▼ To Run the Sample Program `PolicyCreator.java`

1. **Compile the sample code.**

   See "Compiling the Policy Code Samples" on page 76 above.

2. **Set the environment variable LD_LIBRARY_PATH.**

   On Solaris, add `/usr/lib/mps/secv1` to `LD_LIBRARY_PATH`.

   On Linux, add `/opt/sun/private/lib` to `LD_LIBRARY_PATH` .

   On HP-UX, add `/opt/sun/private/lib` to the environment variable `SHLIB_PATH`.

3. **In the administration console, go to Access Control > *root_realm*> Services > Policy Configuration.**

4. **Under "Selected Dynamic Attributes," add the following as the two dynamic attributes to be retrieved as part of the Policy Decision:**

   - `uid`
   - `cn`

5. **Set the following properties in the `PolicyEvaluation.properties` file:**

`pe.realmname`	`DN` of the root realm.
`pe.username`	`UserId` to authenticate as.
`pe.password`	Password to use to authenticate.

6 Run the `gmake createPolicies` command. (On Windows, `make.bat createPolicies`.)

```
gmake createPolicies .
```

Use the administration console to verify that the policies `policy1` and `refpolicy1` are added to Access Manager.

◆ ◆ ◆ CHAPTER 4

Using the JAAS Authorization Framework

Previous versions of Access Manager (Identity Server 6.0 and 6.1) provided custom policy APIs to define and evaluate access policies. This model provided centralized management of policies in its own policy store, the Sun ONE or Java Enterprise System (JES) Directory Server. In Sun Java™ System Access Manager 6.2 and beyond, the authorization segment of the Java Authentication and Authorization Service (JAAS) framework is added to the original model. This model is based on JAAS 1.0 and Java 2 Platform, Standard Edition (J2SE) 1.3.1.

Access Manager now bridges the gap between J2SE and Access Manager APIs. In this framework, Access Manager maps its private APIs to JAAS interfaces. This makes it possible for you to use the JAAS interface to access the Access Manager policy framework.

The topics covered in this chapter are:

Overview of JAAS Authorization

JAAS is a set of APIs that enable services to authenticate and enforce access controls upon users. It implements a Java technology version of the standard Pluggable Authentication Module (PAM) framework, and supports user-based authorization. JAAS authorization extends the Java security architecture which uses a security policy to specify what access rights are granted to executing code. That architecture, introduced in the Java 2 platform, is code-based. The permissions are granted based on code characteristics such as where the code is coming from, whether it is digitally signed, and if so, the identity of the signer.

"Overview of JAAS Authorization" on page 95 illustrates a Java security policy. This grants the code in the `am_services.jar` file, located in the current directory, the specified permission. No signer is specified, so it doesn't matter whether the code is signed or not.

EXAMPLE 4–1 Example of a Java Security Policy

```
grant codebase Cfile:./am_services.jar" {
            permission javax.security.auth.AuthPermission
                            "createLoginContext.AMLoginContext";
         };
```

JAAS authorization adds user centric access control that applies control based on what code is running as well as on who is running it.

By default, JAAS comes with a reference implementation of Policy (`com.sun.security.auth.PolicyFile`) which is file-based. This implementation parses the `Java.policy` file `${java.home}/lib/`*security directory* and uses that to direct the associations of permissions to code. You can change the pointer to some other `PolicyFile` implementation or use a combination of files. By default, two files are consulted to evaluate policy. One is `com.sun.security.auth.PolicyFile`, mentioned above, and the other is `.java.policy` as defined in user's home directory.

To make JAAS authorization take place, include a Principal field in the grant statement or statements in your policy file. A Principal field indicates which user executing the code is allowed the designated permissions. The Policy file grant statements can now optionally include one or more Principal fields. Including Principal field in the grant statement indicates that the user represented by the specified Principal, who is executing the specified code, has the designated permissions. See the Principal field example in "Overview of JAAS Authorization" on page 95.

EXAMPLE 4–2 A Policy File Grant Statement

```
grant   codebase "file:./am_services.jar",

        Principal javax.security.auth.XXXprincipal


        "your_user_name@your_domain"  {


   permission java.util.PropertyPermission "java.home", "read";
   permission java.util.PropertyPermission "user.home", "read";
   permission java.io.FilePermission "foo.txt", "read";
};
```

How Policy Enforcement Works

The Java 2 runtime enforces access controls via the `java.lang.SecurityManager` , which is consulted any time untrusted code attempts to perform a sensitive operation (accesses to the local file system, for example). To determine whether the code has sufficient permissions, the `SecurityManager` implementation delegates responsibility to the `java.security.AccessController`, which first obtains an image of the current `AccessControlContext`, and then ensures that the retrieved `AccessControlContext` contains sufficient permissions for the operation to be permitted.

JAAS supplements this architecture by providing the method `Subject.doAs` to dynamically associate an authenticated subject with the current `AccessControlContext`. As subsequent access control checks are made, the `AccessController` can base its decisions upon both the executing code itself, and upon the principals associated with the subject. Access Manager provides support for JAAS authentication, which results in the population of the subject with Principals that represents the user.

EXAMPLE 4–3 The `Subject.doAs` Method

```
public final class Subject {
                ...
                // associate the subject with the current
                // AccessControlContext and execute the action
                public static Object doAs(Subject s,
                                java.security.PrivilegedAction action) { }
           }
```

To illustrate a usage scenario for the `doAs` method, consider a service that authenticates a remote subject, and then performs some work on behalf of that subject. For security reasons, the server should run in an `AccessControlContext` bound by the subject's permissions. Using JAAS, the server can ensure this by preparing the work to be performed as a `java.security.PrivilegedAction` . Then, by invoking the `doAs` method, the server provides both the authenticated subject and the prepared `PrivilegedAction`. The `doAs` implementation associates the subject with the current `AccessControlContext` and then executes the action. When security checks occur during execution, the Java 2 `SecurityManager` queries the JAAS policy, updates the current `AccessControlContext` with the permissions granted to the subject and the executing `codesource`, and then performs its regular permission checks. When the action is completed, the `doAs` method removes the subject from the current `AccessControlContext`, and returns the result back to the caller. "How Policy Enforcement Works" on page 97 illustrates this flow.

EXAMPLE 4–4 Sample Code for `Subject.doAS`

```
public static void main(String[] args) {
        try {
            // Create an SSOToken
            AuthContext ac = new AuthContext("dc=iplanet,dc=com");
            ac.login();
            Callback[] callbacks = null;
            if (ac.hasMoreRequirements()) {
                callbacks = ac.getRequirements();

                if (callbacks != null) {
                    try {
                        addLoginCallbackMessage(callbacks);
                        // this method sets appropriate responses in the callbacks.
                        ac.submitRequirements(callbacks);
                    } catch (Exception e) { }
                }
            }
            if (ac.getStatus() == AuthContext.Status.SUCCESS) {
                  Subject subject = ac.getSubject();
                        // get the authenticated subject
                  FilePermission perm = new FilePermission("/tmp/test","read");

                      Subject.doAs(subject, new PrivilegedExceptionAction() {
                      /* above statement means execute  run() method of the
                                 /* Class PrivilegedExceptionAction()
                          as the specified subject */
                      public Object run() throws IOException  {
                          // if the above run()  was not throwing Exception
                                    /* could have created an instance of PrivilegedAction
                          // instead of  PrivilegedExceptionAction here
                          AccessController.checkPermission(perm);
                            File = new File("/tmp/test");
                            return null;
                      }
                  });
             }
   }
```

In this example, the `AccessController` is checking the application's current policy implementation. If any permission defined in the policy file implies the requested permission, the method will simply return; otherwise an `AccessControlException` will be thrown. The check is actually redundant in this example, because the constructor for the default File implementation performs the same check. The sample is meant to illustrate the flow.

How the JS2E Access Controller Works

`AccessController` works with the `java.security.Policy` implementation to securely process application requests. In JS2E, a typical `checkPermission(Permission p)` method call on the `AccessController` class might result in the following sequence:

1. The `AccessController` invokes the `getPermisisons()` method of the `javax.security.auth` policy passing in the subject and the code source.
2. The `getPermissions()` method returns a `PermissionCollection` class instance, which represents a collection of same types of permissions.
3. The `elements()` method of the returned `PermissionCollection` gets called, which returns an enumeration of the permissions held in this `PermissionCollection`.
4. For each of the permissions returned in the enumeration (in step 3), the `perm.newPermissionCollection()` method gets called to obtain the `PermissionCollection` used to store the permission.
5. `PermissionCollection.add(perm)` gets called by the J2SE internal code to store the permission in its `PermissionCollection`.
6. The `AccessController` calls the `implies(Permission p)` method of the `PermissionCollection` returned in step 2.
7. Once the `implies()` of `PermissionCollection` is called, it in turn triggers the calling of `implies(Permission p)` of the individual permission objects contained in the `PermissionCollection` . These methods return `true` if the current permission object in the collection implies the specified permission; the methods return `false` the current permission object in the collection does not imply the specified permission. This outcome is implementation dependent and can be changed.

JAAS Authorization in Access Manager

Access Manager provides a custom implementation of the JAAS `javax.security.auth.Policy` . The customized implementation leverages the J2SE access controller and security manager to provide policy evaluation for all Access Manager related permissions. The customized implementation also falls back on the J2SE default Policy implementation `com.sun.security.auth.PolicyFile` for access to system level resources. Access Manager policy does not control access to `com.sun.security.auth.PolicyFile.`

Access Manager uses both JAAS and J2SE's file-based policy for all the resources for which Access Manager does not provide access control. For Access Manager resources such as URLs and so forth, new policy and permissions are defined. This model leverages the best of JAAS and the best of J2SE in one solution. It uses the JAAS framework for its default access control where needed, and then enhances the framework to incorporate the Access Manager policy evaluation. In this way, you can use the Access Manager policy implementation to make policy evaluations pertaining to Access Manager policies, but revert back to the default method of controlling access to resources not under Access Manager control.

Custom APIs

Access Manager provides the following custom APIs:.

- `Package com.sun.identity.policy.jaas`

  This package includes classes for performing policy evaluation against Access Manager using JAAS (Java Authentication and Authorization) framework.

- `ISPermission`

  This class provides the support for JAAS Authorization service. It is a new JAAS Permission which extends the Permission class and is defined to evaluate permission against the Access Manager policy framework.

- `ISPolicy`

  This is an implementation of abstract class `javax.security.auth.Policy` for representing the system security policy for a Java application environment. It performs policy evaluation against the Access manager policy service instead of against the default file-based `PolicyFile`.

For a comprehensive listing of related APIs, see the Javadoc in the following directory: *AccessManager-base*`/SUNWam/docs`.

User Interface

The user interface for entering permissions and policy is the Access Manager administration console which works with the policy administration API. Once the policy is defined, the evaluation is done using the J2SE architecture and enhanced policy implementation.

`ISPermission` covers the case when additional policy services are defined and imported, provided they only have boolean action values. In fact boolean evaluation is all that can be done using JAAS since JAAS permissions have a boolean result.

Enabling the JAAS Authorization Framework

You enable the JAAS authorization framework by resetting policy. Use the `Policy.setPolicy(Policy)` API to reset policy during run time. In "Enabling the JAAS Authorization Framework" on page 100, `Policy.setPolicy(com.sun.identity.policy.jaas.ISPolicy)` resets the policy. In this example, the client application wants to use JAAS authorization API to communicate with the Access Manger and to perform policy evaluation. Access Manager provides the support needed to use Access Manager policy so that policy can be defined through the new `ISPermission`.

EXAMPLE 4–5 Sample JAAS Authorization Code

```
public static void main(String[] args) {
       try {
           // Create an SSOToken

          AuthContext ac = new AuthContext("dc=iplanet,dc=com");
           ac.login();
           Callback[] callbacks = null;
           if (ac.hasMoreRequirements()) {
               callbacks = ac.getRequirements();

               if (callbacks != null) {
                   try {
                       addLoginCallbackMessage(callbacks);
                   // this method sets appropriate responses in the callbacks.
                       ac.submitRequirements(callbacks);
                   } catch (Exception e) { }
               }
           }
           if (ac.getStatus() == AuthContext.Status.SUCCESS) {
                 Subject subject = ac.getSubject();
                           // get the authenticated subject

                   Policy.setPolicy(new ISPolicy()); // change the policy to  our own Policy


                   ISPermission perm = new ("iPlanetAMWebAgentService",

                       "http://www.sun.com:80", "GET");
                 Subject.doAs(subject, new PrivilegedExceptionAction() {
                     /* above statement means execute  run() method of the
                                /* Class PrivilegedExceptionAction()
                        as the specified subject */
                     public Object run() throws Exception {
                         AccessController.checkPermission(perm);
                           // the above will return quietly if the Permission
                                       //  has been granted
                           // else will throw access denied
                           // Exception, so if the above highlighed ISPermission
                                       // had not been granted, this return null;
                     }
                  });
            }
  }
```

EXAMPLE 4–5 Sample JAAS Authorization Code *(Continued)*

◆ ◆ ◆ CHAPTER 5

Using Session Service APIs

The Session Service component of the Sun Java™ System Access Manager 7.1 enables single sign-on (SSO) functionality. In a single sign-on session, a user authenticates or logs in to a protected resource once. Until the user logs out, the user can access a number of other protected resources without having to present credentials again. For detailed information about how the Session Service and SSO work, see Chapter 2, "User Session Management and Single Sign-On," in *Sun Java System Access Manager 7.1 Technical Overview*.

This chapter describes the Session Service Java APIs, and related sample code that comes with Access Manager. Topics included in this chapter are:

About the Single Sign-On Java APIs

Once a user has successfully authenticated to Access Manager, the user object uses browser cookies or URL query parameters to carry a Session ID from one application to the next. Each time the user requests access to a protected application, the new application must verify the user's identity. For example, a user successfully authenticates to the application at `http://orgA.company.com/Store`, and then later tries to access `http://orgA.company.com/UpdateInfo`, a service that is SSO-enabled. The `UpdateInfo` application does not ask for the user to present credentials. Instead, the application uses the Session APIs and the user session to determine if the user is already authenticated. If the Session methods determine that the user has already been authenticated and that the session is still valid, then the `UpdateInfo` application allows the user access to its data and operations. If the user is not already authenticated, or if the session is no longer valid, then the `UpdateInfo` application prompts the user to present credentials a second time. The SSO APIs can also be used to create or destroy a `SSOToken`, or to listen for `SSOToken` events.

Using the SSO Code Samples

Access Manager provides the following code samples that demonstrate how you can use the Single Sign-On APIs. These samples are in the form of either standalone Java application or Java servlets.

`SDKCommandLineSSO.java`	Standalone Java program. Creates a new SSO token given a valid SSO token id. Input: Token id. Output: Basic SSO token information.
`CommandLineSSO.java`	Standalone Java program. Demonstrates the usage of retrieving the user profile given the correct user credentials. Input: Organization name (in DN format). Output: User profile attributes.
`SSOTokenSample.java`	Standalone Java program. Serves as a basis for using SSO API. It demonstrates creating an SSO token and calling various methods from the token including getting/setting the session properties. Input: Token id. Output: Basic SSO token information and session properties.
`SDKSampleServlet.java`	Java Servlet. Demonstrates the usage of retrieving the user profile given the valid cookie set in the browser. Input: None, but require AM session cookie set in the browser. Output: SSO token information and user profile attributes.
`SSOTokenSampleServlet.java` `SampleTokenListener.java`	Java Servlet.

Given the valid cookie sent in the browser, these serve as the basis for using the SSO API. Demonstrates use of the of Session Notification Service as well as getting and setting session properties.

Input: None. Requires Access Manager session cookie set in the browser.

Output: Basic SSO token information and session properties.

Running SSO Code Samples on Solaris

On the Solaris platform, you can run the sample programs in one of the following ways:

▼ To Run a Sample Program from the Access Manager Server

1 Set the environment variables.

The following environment variables are used to run the `make` command. You can also set these variables in the Makefile which is in the same directory as the sample files.

`BASE`	Specify the directory where the Access Manager Server is installed.
`CLASSPATH`	Specify the directory where all the `.JAR` files are installed. Example: *AccessManager-base*`/SUNWam/lib`
`JAVA_HOME`	Specify the JDK version your are using. The version must be JDK 1.3.1 or higher.
`BASE_CLASS_DIR`	Specify the directory where you will keep the sample compiled classes.
`JAR_DIR`	Specify the directory where the `.JAR` of the sample classes will be created. The default is the current directory.

2 In the directory *AccessManager-base*`/SUNWam/samples/sso`**, run the** `gmake` **command.**

3 From the directory `JAR_DIR`**, copy the file** `SSOSample.jar` **to the directory** *AccessManager-base*`/SUNWam/lib`**.**

4 Update the web container classpath.

a. Create a web server admin password file.

```
echo "wadm_password=<WS_ADMINPASSWD>" > /tmp/ws70adminpasswd
```

b. Use `wadm get-jvm-prop` to retrieve the current classpath for the Web Server instance.

```
ORIGCLASSPATH=‘$WADM get-jvm-prop --user=$WS_ADMIN
--password-file=/tmp/ws70adminpasswd--host=$WS_HOST --port=$WS_ADMINPORT
--config=$WS_CONFIG class-path-suffix‘
```

c. Use `wadm set-jvm-prop` to add the `SSOSample.jar` to the Web Server instance's classpath.

```
$WADM set-jvm-prop --user=$WS_ADMIN --password-file=/tmp/ws70adminpasswd
--host=$WS_HOST --port=$WS_ADMINPORT --config=$WS_CONFIG class-path-suffix=
"$ORIGCLASSPATH:AccessManager-base/SUNWam/lib/SSOSample.jar"
```

5 Register the Sample servlet.

a. In the file *WebContainer-base*`/https-`*host.domain*`/web-app/`*SERVICES_DEPLOY_URI*`/WEB-INF/web.xml`, insert the following lines immediately after the last `</servlet>` tag:

```
<servlet>
         <servlet-name>SSOTokenSampleServlet</servlet-name>
       <description>SSOTokenSampleServlet</description>
       <servlet-class>SSOTokenSampleServlet</servlet-class>
       </servlet>
```

b. Insert the following lines immediately after the last `</servlet-mapping>` tag.

```
<servlet-mapping>
       <servlet-name>SSOTokenSampleServlet</servlet-name>
       <url-pattern>/SSOTokenSampleServlet</url-pattern>
       </servlet-mapping>
```

6 Restart the Access Manager server.

7 Log in to the Access Manager console.

To execute `SSOTokenSampleServlet`, you must be authorized to access that resource. If you do not have authorization, the request will be denied. See the instructions for setting policy in the Administration Guide.

8 Use a browser to access the following URL:

protocol`://`*host*`:`*port*`/`*SERVICES-DEPLOY-URI*`/SSOTokenSampleServlet`

The default value of `SERVICES-DEPLOY-URI` is `amserver`.

The host name must be a fully qualified name. Your sample program should display the output in the browser.

▼ To Run a Sample Program on a Remote Client

Before You Begin Install the Access Manager Client APIs in a web container and perform the following steps. In the following example, Sun Java System Web Server is installed in a directory named `iws`, and the Access Manager client APIs are installed in a directory named `opt`. For information on installing the Client APIs, see Chapter 1, "Using the Client SDK."

1. **In the directory** *AccessManager-base*`/SUNWam/samples/sso`**, run the** `gmake` **command.**

2. **Be sure that the following are included in the Web Server classpath in the** `server.xml` **file:**

   - `/opt/SUNWam/samples/sso/SSOSample.jar`
   - `/opt/SUNWam/lib/am_sdk.jar`
   - `/usr/share/lib/mps/secv1/jss4.jar`
   - `/opt/SUNWam/lib/jaxp.jar`
   - `/opt/SUNWam/lib/dom.jar`
   - `/opt/SUNWam/lib/xercesImpl.jar`
   - `/opt/SUNWam/lib/jaas.jar` (Add this only if you are using a JDK version lower than JDK1.4)
   - `/opt/SUNWam/locale`and `/opt/SUNWam/lib` directories

3. **Include** `java.protocol.handler.pkgs=com.iplanet.services.comm` **as an argument to be passed into the Web Server virtual machine (VM).**

   ```
   $WADM create-jvm-options --user=$WS_ADMIN --password-file=/tmp/ws70adminpasswd
   --host=$WS_HOST --port=$WS_ADMINPORT --config=$WS_CONFIG --
   -Djava.protocol.handler.pkgs=com.iplanet.services.comm
   ```

4. **Restart Sun Java System Web Server.**

   If the Access Manager server is running with the Secure Socket Layer (SSL) protocol enabled, you may need to add the following line to the `AMConfig.properties` file for testing purposes:

   ```
   com.iplanet.am.jssproxy.trustAllServerCerts=true
   ```

   This property tells the SSL client in the Client APIs to trust all certificates presented by the servers. Adding this property enables you test the SSL connection without having the root CA for your test certificate installed on the this client. Without this property configured, you must install the SSL server `rootCA` certificate in client trust database, and then make sure that the following properties in `AMConfig.properties` are set to the same values:

   - `com.iplanet.am.admin.cli.certdb.dir`
   - `com.iplanet.am.admin.cli.certdb.prefix`
   - `com.iplanet.am.admin.cli.certdb.passfile`

▼ To Run the Sample Code

1 **In the** `/opt/SUNWam/samples/sso` **directory, run the** `gmake` **command.**

This compiles the samples and creates the necessary JAR files.

2 **Register the sample servlet.**

a. **In the file** *WebServer-base*`/https-`*hostName.domainName*`.com/is-web-apps/services/WEB-INF/web.xml`, **insert the following lines immediately after the last** `</servlet>` **tag.**

```
<servlet>
    <servlet-name>SSOTokenSampleServlet</servlet-name>
    <description>SSOTokenSampleServlet</description>
    <servlet-class>SSOTokenSampleServlet</servlet-class>
    </servlet>
```

b. **Insert the following lines immediately after the last** `</servlet-mapping>` **tag.**

```
<servlet-mapping>
    <servlet-name>SSOTokenSampleServlet</servlet-name>
    <url-pattern>/SSOTokenSampleServlet</url-pattern>
    </servlet-mapping>
```

3 **Restart the web container where the Access Manager Client APIs are installed.**

4 **Log in to the Access Manager server.**

5 **To Invoke the servlet, use a browser to go to the following URL:**

`http://amsdk-`*server.sub.domain*`/servlet/SSOTokenSampleServlet`

The `SSOTokenSampleServlet` servlet validates the session and prints out all relevant session information. You may have to reload the URL (Shift + Reload Button) to see updated information.

6 **Log out of the Access Manager server.**

Because no log out link exists in the sample servlet, you must use a browser to access the Access Manager server log out URL. Example:
`https://`*hostName.domainName*`.com/amserver/UI/Logout`

7 **To verify that the client SSOtoken is no longer valid, invoke the servlet a second time.**

Use a browser to go to the following URL:

`http://amsdk-`*server.sub.domain*`/servlet/SSOTokenSampleServlet`

This time, a session exception occurs. Reload the URL to see the updated information.

To Run a Sample Program on the Remote Client Command Line

Before You Begin You must install the Access Manager Client APIs before you can run a sample program on the remote client command line. For more information on using the Client APIs, see Chapter 1, "Using the Client SDK."

When you run a single sign-on (SSO) program from the command line, your application is not running in a web container, but your application must have access to the cookies from the web container HTTP requests. Your application must extract the Access Manager cookie from the request, and then pass the string value of the cookie into the `createSSOToken` method. Because notifications are only supported in a web container, and because your application is not running in a web container, notifications are not supported in this sample.

1. **In the directory** *AccessManager-base*`/SUNWam/samples/sso`**, run the** `gmake` **command.**
2. **Modify the script** *AccessManager-base*`/SUNWam/samples/sso/run` **to specify the sample program that you want to test.**

   For example, to run `SDKCommandLineSSO.java`, in the last line in the script, replace `CommandLineSSO` with `SDKCommandLineSSO`. The result looks like this:

   ```
   ${JAVA_EXEC} -Xbootclasspath ...SDKCommandLineSSO $@
   ```

3. **If you are using a JDK version lower than JDK1.4, add the following to the classpath:**

   ```
   /opt/SUNWam/lib/jaas.jar
   ```

4. **If SSL is enabled, in the script** *AccessManager-base*`/SUNWam/samples/sso/run`**, add the following VM argument when executing your Java code:**

   ```
   java.protocol.handler.pkgs=com.iplanet.services.comm
   ```

To Test the Command Line

To test the command line you can run the servlet test above, cut and paste the cookie value and pass it in as the token value.

1. **Use a browser to access the following URL:**

   ```
   http://test-server.sun.com:80/amserver/SSOTokenSampleServlet
   ```

   The following output is displayed:

   ```
   SSOToken host name: 123.123.123.123 (Your server's ip address)
       SSOToken Principal name: uid=amAdmin,ou=People,dc=example,dc=com
   ```

```
Authentication type used: LDAP
IPAddress of the host: 123.123.123.123 (Your server's ip address)
The token id is AQIC5wM2LY4Sfcwbdp3gWuB38NA26klnTJlLPknN8t0fPVY=
Property: Company is - Sun Microsystems
Property: Country is - USA
SSO Token Validation test Succeeded
```

2 **In the** *AccessManager-base*`/SUNWam/samples/sso` **directory, execute the** `run` **command:**

```
run AQIC5wM2LY4Sfcwbdp3gWuB38NA26klnTJlLPknN8t0fPVY=
```

The following result is displayed:

```
SSO "AQIC5wM2LY4Sfcwbdp3gWuB38NA26klnTJlLPknN8t0fPVY="
    SSOToken host name: 123.123.123.123 (Your server's ip address)
    SSOToken Principal name: uid=amAdmin,ou=People,dc=example,dc=com
    Authentication type used: LDAP
    IPAddress of the host: 123.123.123.123 (Your server's ip address)
```

Developing Non-Web Based Applications

Access Manager provides the SSO APIs primarily for web-based applications although the APIs can be extended to any non-web-based applications with limitations. When developing non-web-based applications, you can use the SSO APIs in one of two ways:

- The application must obtain the Access Manager cookie value and pass it into the SSO client methods to get to the session token. The method used for this process is application-specific.
- You can use command-line applications such as `amadmin`. In this case, session tokens can be created to access the Directory Server directly. There is no session created, making the Access Manager access valid only within that process or VM.

◆ ◆ ◆ CHAPTER 6

Writing Log Operations

Sun Java™ System Access Manager 7.1 provides a Logging Service for recording information such as user activity, traffic patterns, and authorization violations. The Access Manager Logging APIs enable external applications to take advantage of the Logging Service.

For information about how the Logging Service works and what it logs, see Chapter 6, "Logging and the Java Enterprise System Monitoring Framework," in *Sun Java System Access Manager 7.1 Technical Overview*. This chapter describes how to use the Logging APIs to write log operations and customize logging plug-ins. Topics in this chapter include:

About the Logging Samples

Access Manager provides two comprehensive Logging example programs in the *AccessManager-base* `/SUNWam/samples/logging` directory. `LogSample.java` is a log-writing program, and `LogReaderSample.java` is a log-reading program. The `logging` directory also includes the `Makefile` for compiling and scripts to facilitate running the programs.

Writing LogRecords To A Log File or Table

`LogSample.java` takes several command-line arguments, authenticates with the Access Manager server, creates a `LogRecord`, then logs the log record to the specified log file or table. The Access Manager Logging Service determines whether the log records go to a flat file or to a relational database management system (RDBMS), according to the service configuration. The following example command line uses the `LogSample` script:

```
./RunSample -o dc=iplanet,dc=com -u amadmin -p mypassword -n mylog \
            -m "my message to log in mylog" -l user1 -w user1password
```

In `LogSample.java`, the command-line arguments are read. The following arguments are used to acquire the `SSOToken` that is specified in invoking the `LogRecord(loglevel, message, token)` method:

`-o` organization name

`-u` userID

`-p` userID password

The Logging Service extracts other pieces of information from this `userID` `SSOToken` when processing the `LogRecord` request. Ideally, the `userID` specified is the user who is the subject of the record being logged. The `-m` (message) argument is also used in the `LogRecord` call.

```
userToken =getSessionToken(orgname, args[userSID], args[userPWD]);
logRecord = new LogRecord(java.util.logging.Level.INFO, args[message], userToken);
logRecord.addLogInfo("ModuleName", "MyModule");
```

`MyModule` is added as the `ModuleName` property is added to the `LogRecord` using the `addLogInfo()` call. The `-n` (log name) argument is used in the `Logger.getLogger(logname)` call. The `-l` (logged by userID) and `-w` (logged by userID's password) are used to get the `SSOToken` specified in the `logger.log(logRecord, loggedByToken)` call. Where the `userID` associated with the `LogRecord` `SSOToken` is usually the subject of the log record, the `userID` associated with the `log()` `SSOToken` is the user doing the logging. In the actual log file, the values for the log record fields come from the following parameters:

time added by the Logging Service, and is taken from the Access Manager system clock when the LogRecord is instantiated.

Data The `message` as specified in the `LogRecord()` call. In `LogSample.java`, the value after the `-m` option: `my message to log in mylog`.

ModuleName The value specified for the `ModuleName` property (or `LogConstants.MODULE_NAME` property in the `addLogInfo()` call. If no value is specified, this field will read:`Not Available` .

MessageID	The value specified for the`MessageID` property (or `LogConstants.MESSAGE_ID` property in an `addLogInfo()` call. If no value is specified, this field will read: `Not Available`. `LogSample.java` does not add a value for this property.
Domain	The value for this field is extracted from the `SSOToken` specified in the `LogRecord()` call. This corresponds to the subject, userID's domain, or organization.
ContextID	The value for this field is extracted from the `SSOToken` specified in the `LogRecord()` call.
LogLevel	The value specified in the `LogRecord()` call. In `LogSample.java`, the value is `java.util.logging.Level.INFO` (`INFO` in the log file).
LoginID	The value for this field is extracted from the `SSOToken` specified in the `LogRecord()`call. For example, the value can be the `DN` for the `userID` specified in the `-u` command-line option.
IPAddr	The value for this field is extracted from the `SSOToken` specified in the `LogRecord()` call.
LoggedBy	The value for this field is extracted from the `SSOToken` specified in the `logger.log()` call. For example, the value can be the `DN` for the `userID` specified in the `-l` command-line option.
HostName	The value for this field is extracted from the `SSOToken` specified in the `LogRecord()` call. The value is the host name that corresponds to the address in the `IPAddr` field, if it can be resolved.

Reading LogRecords From A Log File or Table

The log writing sample program `LogSample.java` is fairly straightforward in the way the program writes a single record to a file or table as determined by the Logging Service's configuration. In contrast, the log reading sample program is more complex because you can specify that queries are applied to multiple files or tables.

Caution – Log files and tables in particular can become very large. If you specify multiple logs in a single query, create queries that are very specific, or limited in the number of records to return, or both specific and limited. If a large number of records are returned, the Access Manager resource limits (including those of the hosting system) may be exceeded.

`LogReaderSample.java` requires three command-line arguments which are used to authenticate with the Access Manager server. If you specify a log name, then the sample

becomes a single-log reading application. If you don't specify a log name, reading from multiple logs is allowed. Reading from multiple logs does not preclude reading from a single log. Reading from multiple logs is useful when the exact log names available are unknown. The log reading sample is also very interactive. The following command-line example uses the `LogReaderSample` script:

```
./RunLogReader -o dc=iplanet,dc=com -u amadmin -p mypassword
```

In `LogReaderSample.java`, the command-line arguments are read. The following arguments are used to obtain the `SSOToken` that is specified in invoking the various `LogReader.read()` methods:

`-o` organization name

`-u` `userID`

`-p` `userID` password

The LDAP login utility `ldapLogin()` is provided in a separate file, `LogSampleUtils.java`.

Next, the Logging Service configuration is read to determine, for example, whether file or database logging is specified and which log fields are logged.

```
manager.readConfiguration();
String logStorageType = manager.getProperty(LogConstants.BACKEND);
```

Depending on whether Access Manager Logging Service is logging to a file or to a database, when the `LogReader.getSize()` method is invoked on a particular log name, `LogReader.getSizeUnits()` will return either `LogConstants.NUM_BYTES` or `LogConstants.NUM_RECORDS`. For example:

```
 i3 = LogReader.getSizeUnits();
```

The `LogConstants.LOG_FIELDS` property specifies which log fields have been specified for inclusion in the log record. For example:

```
String selFldsStr = manager.getProperty(LogConstants.LOG_FIELDS);
```

The `time` and `Data` fields are mandatory, thus they do not appear in the Logging Service list. They must be explicitly added to the Set of Fields to Retrieve.

```
    StringTokenizer stoken = new StringTokenizer(selFldsStr, ", ");
                    String [] sFields = new String[stoken.countTokens() + 3];
                    Set allFields = new HashSet();

                    allFields.add("time");
                    allFields.add("data");
```

To get the Set of Log Names Available to read and their sizes:

```
    Set filesThereAre = LogReader.getLogNames();
for (Iterator it=filesThereAre.iterator(); it.hasNext(); ) {
    String fileName = (String)it.next();
    long li = 0;
    try {
        li = LogReader.getSize(fileName);
    } catch (Exception ex) {
        System.out.println("got exception on file " +
                fileName + ". " + ex.getMessage());
    }
    System.out.println (fileOrTable + " " + (i2++) +
        " = " + fileName + " contains " + li + " " +
        sizeUnit + ".");
}
```

`LogReaderSample.java` allows you to select reads on a single or multiple logs. If a log name was specified on the command line with the `-n` option, then you can select from among the following types of reads:

```
1. read all records
2. specify logType
3. specify logType and timeStamp
4. specify logType and logQuery
5. specify logType, timeStamp, and logQuery
6. specify logQuery
```

If no log name was specified on the command line, and you select single log to read, you may select from only a list of pre—configured reports:

```
Single (s) or multiple (m) file/table read: [s]
        What type of audit report to generate:
         1. all records from file/table
         2. authentication successes
         3. authentication failures
         4. login/logout activity
         5. policy allows
         6. policy denies
         7. amAdmin CLI activity
         8. amAdmin console activity
         9. Federation access
        10. Federation errors
        11. Liberty access
        12. Liberty errors
        13. SAML access
```

```
            14. SAML error
                    enter type [1..14]:
```

If you want to read from a selected single log, but specify the `logQuery` settings, do not use the `-n` command-line option. Select multiple log read, and then select the single log from which to read:

```
    Available files:
            file 0 = amAuthentication.access contains 1595 bytes.
            file 1 = amPolicy.access contains 2515 bytes.
            ...
            file 13 = amAuthentication.error contains 795 bytes.

            Single (s) or multiple (m) file/table read: [s] m

            Available files:
            0: amAuthentication.access
            1: amPolicy.access
            ...
            12: amConsole.access-1
            13: amAuthentication.error

            Enter selections (space-separated): 0
            What type of read to use:
                1. read all records
                2. specify logQuery
                     enter type [1 or 2]:
```

The following table provides brief descriptions of the `LogReader.read()` methods.

TABLE 6–1 LogReader.read() Methods

`read(String fileName, Object userCrdential)`	Returns all of the records from the specified log, ignoring the maximum number of records specified in the Logging Service configuration.
`read(String logName, String logType, Object userCrdential)`	Specifies the log name and its suffix (`type`) separately, where the suffix can be `access` or `error`. All records are retrieved from the specified log.
`read(String logName, String logType, String timeStamp, Object userCrdential)`	Used when reading secure log files. The `timeStamp` is the suffix that appears after the file `logType` (`access` or `error`). All records are retrieved from the specified log.

TABLE 6–1 LogReader.read() Methods *(Continued)*

`read(String logName, String logType, LogQuery logQuery, Object userCrdential)`	Performs a query, as specified by the `logQuery` parameter. The log name and type (`access` or `error`) are also specified.
`read(String logName, String logType, String timeStamp, LogQuery logQuery, Object userCrdential)`	Corresponds to the method described above. Used in the secure logging case.
`read(String logName, LogQuery logQuery, Object userCrdential)`	Performs a query on the specified log.
`read(String logName, Set fileNames, LogQuery logQuery, Object userCrdential)`	Performs a query on the specified Set of Logs.

The `LogQuery`, along with the `QueryElements` that may be specified, are constructed in the `getLogQuery()` routine in `LogReaderSample.java`.

The following are brief descriptions of the `LogQuery` constructors.

`LogQuery()`
Creates a new `LogQuery` object with the following default values:

```
maxRecord =
    LogQuery.MOST_RECENT_MAX_RECORDS
    globalOperand =
    LogQuery.MATCH_ANY_CONDITION
queries = null (QueryElement)
columns = null (columns to return)
sortBy = null (field to sort on)
```

`LogQuery(int max_record)`
Creates a new `LogQuery` object with the following values:

```
maxRecord = max_record
globalOperand =     LogQuery.MATCH_ANY_CONDITION
queries = null (QueryElement)
columns = null (columns to return)
sortBy = null (field to sort on)
```

`LogQuery(int max_Record, int matchCriteria, java.lang.String sortingBy)`
Creates a new `LogQuery` object with the following values:

```
maxRecord = max_Record
globalOperand = matchCriteria
queries = null (QueryElement)
columns = null (columns to return)
sortBy = sortingBy (field to sort on)
```

The `LogQuery` object created with the constructors may be subsequently modified with the following `set*` methods:

- `setColumns(java.util.ArrayList columns)`
- `setGlobalOperand(int no)`
- `setMaxRecord(int value)`
- `setSortingField(java.lang.String fieldName)`

A `LogQuery` may specify a List of `QueryElements`, each containing a value for a field (column) and a relationship. The following sample code queries for all successful authentications in domain `dc=iplanet,dc=com`, and returns the `time`, `Data`, `MessageID`, `ContextID`, `LoginID`, and `Domain` fields, sorted on the `LoginID` field:

```
ArrayList al = new ArrayList();
al.add (LogConstants.TIME);
al.add (LogConstants.Data);
al.add (LogConstants.MESSAGE_ID);
al.add (LogConstants.CONTEXT_ID);
al.add (LogConstants.LOGIN_ID);
al.add (LogConstants.DOMAIN);
LogQuery lq = new LogQuery(LogQuery.ALL_RECORDS,
        LogQuery.MATCH_ALL_CONDITIONS,
        LogConstants.LOGIN_ID);

QueryElement qe1 = new QueryElement(LogConstants.MESSAGE_ID,
        "AUTHENTICATION-105",
        QueryElement.EQ);
lq.addQuery(qe1);

QueryElement qe2 = new QueryElement(LogConstants.DOMAIN,
        "dc=iplanet,dc=com",
        QueryElement.EQ);
lq.addQuery(qe2);
```

`QueryElement` supports the following relationships:

`QueryElement.GT`	Greater than
`QueryElement.LT`	Less than
`QueryElement.EQ`	Equal to
`QueryElement.NE`	Not equal to

`QueryElement.GE`	Greater than or equal to
`QueryElement.LE`	Less than or equal to
`QueryElement.CN`	Contains
`QueryElement.SW`	Starts with
`QueryElement.EW`	Ends with

In the example, assuming that `dc=iplanet,dc=com` is the root domain, changing the `qe2`relationship field to `QueryElement.EW` (Ends with) or `QueryElement.CN` (Contains) changes the query to include all successful authentications in all domains. To read the example query from the `amAuthentication.access log`, assuming the `SSOToken` is in `ssoToken`:

```
String[][] result = new String[1][1];
   result = read("amAuthentication.access", lq, ssoToken);
```

The first record (row 0) contains the field and column names. See the `printResults()` method in `LogReaderSample.java` for a sample display routine.

Compiling Logging Programs

Included with the sample log programs is a `gmake` Makefile which compiles both `LogSample.java` and `LogReaderSample.java`, as well as the utilities module `LogSampleUtils.java`. The item of most interest is the `CLASSPATH` setting.

Executing Logging Programs

The sample standalone log programs include `ksh` scripts. There are considerations for running on Solaris or Linux handled by the scripts, but a few less obvious settings concern whether there is local or remote logging, if database logging is configured, and if Access Manager is configured for SSL. The `LOCAL_LOGGING` shell variable is set to `true` by default. If the logging program is executing on a remote system, using the Access Manager client APIs, then the `LOCAL_LOGGING` shell variable this must be set to `false`. The `LOCAL_LOGGING` setting later determines the setting of the `CONFIGOPTION` variable. When the logging program is running on the same system as the Access Manager server, and logging to a database is configured, then the database JDBC driver must also be included in the `CLASSPATH`. If the Access Manager server is configured for SSL, and the logging program is executing on a remote system using the Access Manager client APIs, be sure that the following parameter is set in the script:

```
-D"java.protocol.handler.pkgs=com.iplanet.services.comm"
```

The certificate database conforming to the Access Manager server container must be provided, and the `com.iplanet.am.admin.cli.certdb.dir` property in the `AMConfig.properties` file must point to the Access Manager server container. For example, for non-production testing to an Access Manager server running in a Application Server 8.1 container, you can copy (assuming default installation of AS 8.1) `/var/opt/SUNWappserver/domains/domain1/config` to the remote system, and `set com.iplanet.am.admin.cli.certdb.dir` to that location. You must also set the following:

```
com.iplanet.am.admin.cli.certdb.prefix=
    com.iplanet.am.admin.cli.certdb.passfile=/etc/opt/SUNWam/config/.wtpass
```

The `.wtpass` file needs to be created. More detailed information about certificates, see the file *AccessManager-base*`/SUNWam/samples/authentication/api/Readme_setup.html` .

Implementing a Remote Logging Application in a Container

If your remote logging application is running in a container such as Sun Java System Application Server or Web Server, at the command line, set the following properties:

```
-Ds1is.java.util.logging.config.class=
            com.sun.identity.log.s1is.LogConfigReader

-DLOG_COMPATMODE=Off

-Djava.util.logging.manager=
            com.iplanet.ias.server.logging.ServerLogManager
```

The `-Djava.util.logging.manager` property occurs in the Java System Web Server `server.xml` file. JVM options are typically added to the `server.xml` file in Java System Web Server, or to the `domain.xml` file in Java System Application Server.

Setting Environment Variables

You must set the following shared library environment variables in the executable for an application that is using the Logging Service. You can determine how to set the variables depending upon three things:

- Whether the application can execute in the local Access Manager server, or executes only a in remote server
- Whether or not you want the Access Manager `LogManager` class to override the native `LogManager class`
- Whether or not SSL is enabled in your deployment

If Client Can Execute in the Local Access Manager Server

When the client application can execute in either the local Access Manager server JVM or in a remote server JVM, choose one of the following two configurations:

- If it is acceptable for the native `LogManager` class to be overridden by the Access Manager `LogManager` class in the JDK1.4 environment, then set the following variables:

```
-D"java.util.logging.manager=com.sun.identity.log.LogManager"
-D"java.util.logging.config.class=com.sun.identity.log.
            s1is.LogConfigReader"
```

- If it is *not* acceptable for the native `LogManager` class to be overridden by the Access Manager `LogManager` class in the JDK1.4 environment, then set the following variables:

```
-DLOG_COMPATMODE=Off
-Ds1is.java.util.logging.config.class=com.sun.identity.log.
            s1is.LogConfigReader
```

If Client Executes Only in a Remote Server

When the client application can execute only in a remote server JVM, choose one of the following two configurations:

- If it is acceptable for the native `LogManager` class to be overridden by the Access Manager `LogManager` class in the JDK1.4 environment, then follow these steps:

  1. Set the following variables:

```
-Djava.util.logging.manager=com.sun.identity.log.LogManager

-Djava.util.logging.config.file=/AccessManager_base/SUNwam/
        lib/LogConfig.properties
```

  2. In `LogConfig.properties`, or in the `logging.properties` file supplied by JDK, set the following properties:

```
iplanet-am-logging-remote-handler=com.sun.identity.log.handlers.
        RemoteHandler

iplanet-am-logging-remote-formatter=com.sun.identity.log.
        handlers.RemoteFormatter

iplanet-am-logging-remote-buffer-size=1

iplanet-am-logging-buffer-time-in-seconds=3600

iplanet-am-logging-time-buffering-status=OFF
```

- If it is *not* acceptable for the native `LogManager` class to be overridden by the Access Manager `LogManager` class in the JDK1.4 environment, then follow these steps:
  1. Set the following variables:

     ```
     -DLOG_COMPATMODE=Off
     ```

     ```
     -Ds1is.java.util.logging.config.file=/AccessManager-base/SUNwam/lib/LogConfig.proper
     ```

  2. In `LogConfig.properties`, or in the `logging.properties` file supplied by JDK, set the following properties:

     ```
     iplanet-am-logging-remote-handler=com.sun.identity.log.
                handlers.RemoteHandler
     iplanet-am-logging-remote-formatter=com.sun.identity.log.
                handlers.RemoteFormatter

     iplanet-am-logging-remote-buffer-size=1

     iplanet-am-logging-buffer-time-in-seconds=3600

     iplanet-am-logging-time-buffering-status=OFF
     ```

The Client APIs use this logging configuration by default. In this case, the Logging API will configure a remote handler for all logs. Access to the Directory Server is not required in this case.

If SSL is Enabled

If SSL is enable and uses JSS for Access Manager, set the following parameter:

```
-D"java.protocol.handler.pkgs=com.iplanet.services.comm"
```

Logging to a Second Access Manager Server

For a `remote` Access Manager server to use another Access Manager server's logging service, set the Logging Service URL in the remote Access Manager server Naming Service to specify the Access Manager server that will be performing the actual logging. User the following form:

`http://`*host*`:`*port*`/`*amserver*`/loggingservice`

Using the Logging Sample Files

The sample files demonstrate how you can use the Access Manager Logging APIs for to log operations. You can execute the samples through the command line. You must have super user privileges to run the `RunSample` and `RunLogReader` programs and to access `AMConfig.properties`.

▼ To Run the Sample Programs on Solaris

1 **In the `Makefile`, `RunSample`, and `RunLogReader` files, set the following variables. The variables may already have been set during installation.**

`AM_HOME`	Set this to refer to the where Access Manager server is installed.
`JAVA_HOME`	Set this variable to your installation of the JDK. The JDK version should be greater than or equal to 1.3.1_06.
`JDK14`	Set this variable to `true` if your `JAVA_HOME` points to JDK 1.4 or newer version else set it to `false`
`LOCAL_LOGGING`	Set this variable to `true` if you are executing this sample at complete Access Manager installation which will perform local logging. If you are executing this sample from a `SUNWamsdk` only install then set it to `false` which will perform remote logging (logging at server side).

2 **Set the `LD_LIBRARY_PATH` as is appropriate for your installation.**

3 **Run the `gmake` command to compile the sample program.**

4 **Run the following `chmod` command:**

```
chmod +x RunSample RunLogReader
```

5 **Run the following command to run the logging sample program:**

`./RunSample [ -o` *organizationName*`] [ -u` *userName* `-p` *userPassword* `]`
`-n` *logName* `-m` *message* `-l` *loggedByUser* `-w` *loggedByUserPassword*

orgName	Name of the organization. This is an optional parameter. If a value is not provided, Access Manager assumes the value to be the root organization.
userName	Name of the user on whose behalf the logging is performed. This is an optional parameter.
userPassword	Password for authenticating the user. This value must be provided if *userName* is provided.

logName	Name of the log file.
message	Message to be logged to the log file.
loggedByUser	Name of the administrator user who is logging the message.
loggedByUserPassword	Password to authenticate the administrator user.

Example:

```
$ ./RunSample -u amadmin -p 11111111 -n testLog.access -m "trying test logging"-l
amadmin -w 11111111
```

6 Run the log reader program by running the following command:

`./RunLogReader -o` *organizationName* `-u` *userName*
`-p` *userPassword* `[-n` *logName*`]`

organizationName	Name of the organization. This is a required parameter.
username	Name of the user who is accessing the log file or table. This is a required parameter.
userpassword	Password to authenticate the user. This is a required parameter.
logName	Name of the log file or table. This parameter is optional. You can select the log file or table when running the program.

Example :

```
$ ./RunLogReader -u amadmin -p 11111111 -o dc=example,dc=com
             -n testLog.access
```

▼ To Run the Sample Programs on Windows 2000

1 In the `make.bat` file, set the following variables:

`BASE`	Set this to refer to the where Access Manager server is installed.
`JAVA_HOME`	Set this variable to your installation of the JDK. The JDK version should be greater than or equal to 1.3.1_06.
`JDK14`	Set this variable to `true` if your `JAVA_HOME` points to JDK 1.4 or newer version. Otherwise, set it to `false`.
`LOCAL_LOGGING`	Set this variable to `true` if you are executing this sample at complete Access Manager installation which will perform local logging. If you are executing this sample from an `SUNWamsdk` only install then set it to `false` which will perform remote logging (logging at server side).

2 **Set the** `LD_LIBRARY_PATH` **as is appropriate for your installation.**

3 **Compile the program by running the** `make` **command.**

4 **Run the sample program by running the** `make run` **command:**

```
make run [-o organizationName]
             [-u userName -p  userPassword]      -n logName
            -m message -l  loggedByUser
                     -wloggedByUserPassword
```

orgName	Name of the organization. This is an optional parameter. If a value is not provided, Access Manager assumes the value to be the root organization.
userName	Name of the user on whose behalf the logging is performed. This is an optional parameter.
userPassword	Password for authenticating the user. This value must be provided if *userName* is provided.
logName	Name of the log file.
message	Message to be logged to the log file.
loggedByUser	Name of the administrator user who is logging the message.
loggedByUserPassword	Password to authenticate the administrator user.

Example:

```
c> make run -u amadmin -p 11111111 -n testLog.access
            -m "trying test logging" -l amadmin -w 11111111
```

Using the Logging SPIs

The Logging SPI are Java packages that can be used to develop plug-ins for customized features. The SPI are organized in the `com.sun.identity.log.spi` package. For more information, see the *Sun Java System Access Manager 7.1 Java API Reference*.

Log Verifier Plug-In

If secure logging is enabled, the log files are verified periodically to detect any attempt of tampering. If tampering is detected, the action taken can be customized by following the steps.

▼ To Customize Actions to be Taken in Secure Logging

1. **Implement the `com.sun.identity.log.spi.IVerifierOutput` interface with the desired functionality.**
2. **Add the implementing class in the classpath of Access Manager.**
3. **Modify the property `iplanet-am-logging-verifier-action-class` in the `/etc/opt/SUNWam/config/xml/amLogging.xml` file with the name of the new class.**

Log Authorization Plug-In

The Logging Service enables you to plug in a class that will determine whether a `LogRecord` is logged or discarded. The determination is based on the authorization of the owner of the session token performing the event.

Note – The `IAuthorizer` interface accepts an SSOToken and the log record being written.

There are several ways to accomplish this. The following procedure is one example.

▼ To Implement a Log Authorization Plug-In

1. **Get the applicable role or DN of the user from the `SSOToken` and check it against a pre-configured (or hardcoded) list of roles or users that are allowed access.**

   The administrator must configure a role and assign all policy agents and entities such as applications that can possibly log into Access Manager and into this role.
2. **Instantiate a `PolicyEvaluator` and call `PolicyEvaluator.isAllowed(ssotoken, logname);`.**

▼ To Instantiate a `PolicyEvaluator`

This entails defining a policy XML to model log access and registering it with Access Manager.

1. **Implement the `com.sun.identity.log.spi.IAuthorizer` interface with the desired functionality.**
2. **Add the implementing class in the classpath of Access Manager.**
3. **Modify the property `iplanet-am-logging-authz-class` in the `/etc/opt/SUNWam/config/xml/amLogging.xml` file with the name of the new class.**

◆ ◆ ◆ CHAPTER 7

Client Detection Service

The Sun Java™ System Access Manager 7.1 Authentication Service has the capability of being accessed from many client types, whether HTML-based, WML-based or other protocols. In order for this function to work, Access Manager must be able to identify the client type. The Client Detection Service is used for this purpose. This chapter offers information on the service, and how it can be used to recognize the client type. It contains the following sections:

Overview

The Access Manager Authentication Service has the capability to process requests from multiple browser type clients. Thus, the service can be used to authenticate users attempting to access applications based in HTML, WML or other protocols.

Caution – The Access Manager console though cannot be accessed from any client type except HTML.

The client detection API can be used to determine the protocol of the requesting client browser and retrieve the correctly formatted pages for the particular client type.

Client Detection Process

Since any user requesting access to Access Manager must first be successfully authenticated, browser type client detection is accomplished within the Authentication Service. When a client's request is passed to Access Manager, it is directed to the Authentication Service. Within this service, the first step in user validation is to identify the browser type using the `User-Agent` field stored in the HTTP request.

Note – The `User-Agent` field contains *product tokens* which contains information about the browser type client originating the HTTP request. The tokens are a standard used to allow communicating applications to identify themselves. The format is `software/version library/version`.

The `User-Agent` information is then matched to browser type data defined and stored in the `amClientData.xml` file.

Caution – User-Agent information is defined in `amClientData.xml` but this information is stored in Directory Server under Client Detection Service.

Based on this client data, correctly formatted browser pages are sent back to the client for authentication (for example, HTML or WML pages). Once the user is validated, the client type is added to the session token (as the key `clientType`) where it can be retrieved and used by other Access Manager services. (If there is no matching client data, the default type is returned.)

Note – The `userAgent` must be a part of the client data configured for all browser type clients. It can be a partial string or the exact product token.

Enabling Client Detection

By default, the client detection capability is disabled; this then assumes the client to be of the `genericHTML` type (For example Access Manager will be accessed from a HTML browser). The preferred way to enable the Client Detection Service is to use the Access Manager console and select the option in the Client Detection Service itself. For more information, see the Administration Guide. To enable client detection using the `amClientDetection.xml`, the `iplanet-am-client-detection-enabled` attribute must be set to `true`. `amClientDetection.xml` must then be deleted from Directory Server and reloaded using `amAdmin`. The following procedure illustrates the complete enabling process.

1. **Import client data XML file using the `amadmin` command** */AccessManager-base amadmin_DN* **-w** *amadmin_password* **-t** *name_of_XML_file*

   This step is only necessary if the client data is not already defined in `amClientData.xml`.

2. **Restart Access Manager.**

3. **Login to Access Manager console.**

4. **Go to Service Configuration and click `ClientDetectionproperties`.**

5. **Enable Client Detection.**

6 **Make sure the imported data can be viewed with Access Manager console.**

Click on the Edit button next to the Client Data attribute.

7 **Create a directory for new client type and add customized JSPs.**

Create a new directory in */AccessManager-base*/SUNWam/web-src/services/config/auth/default/ and add JSPs for the new client type. Client Detection Process is a login page written for a WML browser.

```
<?xml version="1.0"?>

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN">
<"http://www.wapforum.org/DTD/wml_1.1.xml">

<!-- Copyright Sun Microsystems, Inc.  All Rights Reserved -->

<wml>
<head>
<meta http-equiv="Cache-Control" content="max-age=0"/>
</head>

 <card id="authmenu" title="Username">
<do type="accept" label="Enter">

<go method="get" href="/wireless">
<postfield name="TOKEN0" value="$username"/>
<postfield name="TOKEN1" value="$password"/>
</go>
</do>
<p>
Enter username:
<input type="text" name="password"/>
</p>
<p>
Enter password:
<input type="text" name="username"/>
</p>
</card>
</wml>
```

Client Data

In order to detect client types, Access Manager needs to recognize their identifying characteristics. These characteristics identify the features of all supported types and are defined in the `amClientData.xml` service file. The full scope of client data available is defined as a schema in `amClientData.xml` . The configured Access Manager client data available for HTML-based browsers is defined as sub-configurations of the overall schema: `genericHTML` and its parent HTML.

Note – Parent profiles (or *styles*, as they are referred to in the Access Manager console) are defined with properties that are common to its configured child devices. This allows for the dynamic inheritance of the parent properties to the child devices making the device profiles easier to mange.

HTML

`HTML` is a base style containing properties common to HTML-based browsers. It might have several branches including web-based HTML (or `genericHTML`), `cHTML` (Compact HTML) and others. All configured devices for this style could inherit these properties which include:

`parentId`	Identifies the base profile. The default value is `HTML`.
`clientType`	Arbitrary string which uniquely identifies the client. The default value is `HTML`.
`filePath`	Used to locate the client type files (templates and JSP files). The default value is `html`.
`contentType`	Defines the content type of the HTTP request. The default value is `text/html`.
`genericHTML`—defines	Client that will be treated as HTML. The default value is `true`. This attribute does not refer to the similarly named generic HTML style.
`cookieSupport`	Defines whether cookies are supported by the client browser. The default value is `true` which sets a cookie in the response header. The other two values could be `False` which sets the cookie in the URL and `Null` which allows for dynamic cookie detection. In the first request, the cookie is set in both the response header and the URL; the actual mode is then detected and set from the subsequent request. Although the Client Detection Service supports a cookieless mode, Access Manager console does not. Therefore, enabling this function will not allow login to the console. This feature is provided for wireless applications and others that will support it.

`CcppAccept-Charset` Defines the character encoding used by Access Manager to send a response to the browser. The default value is `UTF-8`.

genericHTML

`genericHTML`refers to an HTML browser such as Netscape Navigator™, Microsoft™ Internet Explorer, or Mozilla™. As a configured device, inherits properties from the `HTML` style as well as defining its own properties. `genericHTML` properties include the following:

`parentId` Identifies the base profile for the configured device. The default value is `HTML`.

`clientType` An arbitrary string which uniquely identifies the client. The default value is `genericHTML`.

`userAgent` Search filter used to compare/match the user agent defined in the HTTP header. The default value is `Mozilla/4.0` .

`CcppAccept-Charset`

Defines the character encoding set supported by the browser. The default values are :

```
UTF-8;ISO-8859-1;ISO-8859-2;
ISO-8859-3;ISO-8859-4;ISO-8859-5;
ISO-8859-6;ISO-8859-7;ISO-8859-8;
ISO-8859-9;ISO-8859-10;ISO-8859-14;
ISO-8859-15;Shift_JIS;EUC-JP;
ISO-2022-JP;GB18030;GB2312;BIG5;
EUC-KR;ISO-2022-KR;TIS-620;KOI8-R
```

The character set can be configured for any given locale by adding `charset_locale=codeset` where the code set name is based on the Internet Assigned Numbers Authority (IANA) standard.

Client Detection APIs

Access Manager is packaged with Java APIs which can implement the client detection functionality. The client detection APIs are contained in a package named `com.iplanet.services.cdm` . This package provides the interfaces and classes you need to retrieve client properties. The client detection procedure entails defining the client type characteristics and implementing the client detection API within the external application.

The client detection capability is provided by `ClientDetectionInterface`, a pluggable interface (not an API invoked by a regular application). `ClientDetectionInterface` provides a

`getClientType` method. The `getClientType` method extracts the client data from the browser's incoming `HttpRequest`, matches the user agent information and returns the `ClientType` as a string. Upon successful authentication, the client type is added to the user's session token. The `ClientDetectionException` handles any error conditions.

◆ ◆ ◆ CHAPTER 8

Using the Access Manager Utilities

Sun Java™ System Access Manager 7.1 provides scripts to backup and restore data as well as APIs that are used by the server itself or by external applications. This chapter describes the scripts and the APIs. The chapter contains the following sections:

Utility APIs

The utilities package is called `com.iplanet.am.util`. It contains utility programs that can be used by external applications accessing Access Manager. Following is a summary of the utility API and their functions.

AdminUtils

This class contains the methods used to retrieve the `TopLevelAdmin DN` and password. The information comes from the server configuration file, `serverconfig.xml`, located in *AccessManager-base*`/SUNWam/config/ums`.

AMClientDetector

The `AMClientDetector` interface executes the Client Detection Class configured in the Client Detection Service to get the client type.

AMPasswordUtil

The `AMPasswordUtil` interface has two purposes:

- Encrypting and decrypting any string.

- Encrypting and decrypting special user passwords such as the password for `dsameuser` or proxy user.

  Any remote application using this utility should have the value of the `AMConfig` property `am.encryption.pwd` copied to a properties file on the client side. This value is generated at installation time and stored in `/etc/opt/SUNWam/config/AMConfig.properties` on `Solaris`, `/etc/opt/sun/identity/AMConfig.properties` on `Linux`.

Debug

The Debug utility allows an interface to file debug and exception information in a uniform format. It supports different levels of information (in the ascending order): `OFF`, `ERROR`, `WARNING`, `MESSAGE` and `ON`. A given debug level is enabled if it is set to at least that level. For example, if the debug state is `ERROR`, only errors will be filed. If the debug state is `WARNING`, only errors and warnings will be filed. If the debug state is `MESSAGE`, everything will be filed. `MESSAGE` and `ON` are the same level except `MESSAGE` writes to a file, whereas `ON` writes to `System.out`.

Note – Debugging is an intensive operation and can hurt performance. Java evaluates the arguments to `message()` and `warning()` even when debugging is turned off. It is recommended that the debug state be checked before invoking any `message()` or `warning()` methods to avoid unnecessary argument evaluation and maximize application performance.

Locale

This class is a utility that provides the functionality for applications and services to internationalize their messages.

SystemProperties

This class provides functionality that allows single-point-of-access to all related system properties. First, the class tries to find `AMConfig.class`, and then a file, `AMConfig.properties`, in the CLASSPATH accessible to this code. The class takes precedence over the flat file. If multiple servers are running, each may have their own configuration file. The naming convention for such scenarios is `AMConfig_serverName`.

ThreadPool

`ThreadPool` is a generic thread pool that manages and recycles threads instead of creating them when a task needs to be run on a different thread. Thread pooling saves the virtual machine the work of creating new threads for every short-lived task. In addition, it minimizes the overhead

associated with getting a thread started and cleaning it up after it dies. By creating a pool of threads, a single thread from the pool can be reused any number of times for different tasks. This reduces response time because a thread is already constructed and started and is simply waiting for its next task.

Another characteristic of this thread pool is that it is fixed in size at the time of construction. All the threads are started, and then each goes into a wait state until a task is assigned to it. If all the threads in the pool are currently assigned a task, the pool is empty and new requests (tasks) will have to wait before being scheduled to run. This is a way to put an upper bound on the amount of resources any pool can use up. In the future, this class may be enhanced to provide support growing the size of the pool at runtime to facilitate dynamic tuning.

Password API Plug-Ins

The Password API plug-ins can be used to integrate password functions into applications. They can be used to generate new passwords as well as notify users when their password has been changed. These interfaces are `PasswordGenerator` and `NotifyPassword`, respectively. They can be found in the `com.sun.identity.password.plugins` package.

Note – The Access Manager Javadocs can be accessed from any browser by copying the complete *AccessManager-base*`/SUNWam/docs/` directory into the *AccessManager-base* `/SUNWam/public_html` directory and pointing the browser to `http://` *AcceessManager-HostName.domain_name* `:`*port*`/docs/ index.html`.

There are samples (which include sample code) for these API that can be accessed from the Access Manager installation. They are located in *AccessManager-base* `/SUNWam/samples/console`. They include:

Notify Password Sample

This sample details how to build a plug-in which an administrator can define their own method of notification when a user has reset a password. Instructions for this sample are in the `Readme.txt` or `Readme.html` file located in *AccessManager-base*`/SUNWam/samples/console/NotifyPassword`.

Password Generator Sample

This sample details how to build a plug-in which an administrator can define their own method of random password generation when a user's password is reset using the Password Reset Service. Instructions for this sample are in the `Readme.txt` or `Readme.html` file located in *AccessManager-base*`/SUNWam/samples/console/PasswordGenerator`.

◆ ◆ ◆ CHAPTER 9

Enabling the Access Manager Notification Service

Sun Java™ System Access Manager 7.1 Notification Service allows for session notifications to be sent to remote web containers. It is necessary to enable this service for use by SDK applications running remotely from the Access Manager server itself. This chapter explains how to enable a remote web container to receive the notifications. It contains the following sections:

Overview

The Notification Service allows for session notifications to be sent to web containers that are running the Access Manager SDK remotely. The notifications apply to the Session, Policy and Naming Services only. In addition, the remote application must be running in a web container. The purpose of the notifications would be:

- To sync up the client side cache of the respective services.
- To enable more real time updates on the clients. (Polling is used in absence of notifications.)
- No client application changes are required to support notifications.

Note that the notifications can be received only if the remote SDK is installed on a web container.

Enabling The Notification Service

Following are the steps to configure the remote SSO SDK to receive session notifications.

▼ To Receive Session Notifications

1. **Install Access Manager on Machine 1.**

2. **Install Sun Java System Web Server on Machine 2.**

3. **Install the `SUNWamsdk` on the same machine as the Web Server.**

   For instructions on installing the Access Manager SDK remotely, see the *Sun Java Enterprise System 5 Installation Guide for Unix*.

4. **Ensure that the following are true concerning the machine where the SDK is installed.**

   a. **Ensure that the right access permissions are set for the /** *remote_SDK_server*/`SUNWam/lib` **and /** *remote_SDK_server* **/** `SUNWam/locale` **directories on the server where the SDK is installed.**

      These directories contains the files and jars on the remote server.

   b. **Ensure that the following permissions are set in the Grant section of the** `server.policy` **file of the Web Server.**

      `server.policy` is in the `config` directory of the Web Server installation. These permissions can be copied and pasted, if necessary:

      ```
      permission java.security.SecurityPermission
      "putProviderProperty.Mozilla-JSS"

      permission java.security.SecurityPermission "insertProvider.Mozilla-JSS";
      ```

   c. **Ensure that the correct classpath is set in** `server.xml`**.**

      `server.xml` is also in the `config` directory of the Web Server installation. A typical classpath would be:

      ```
      <JAVA javahome="/export/home/ws61/bin/https/jdk"
      serverclasspath="/export/home/ws61/bin/https/jar/webserv-rt.jar:
      ${java.home}/lib/tools.jar:/export/home/ws61/bin/https/jar/webserv-ext.jar:
      /export/home/ws61/bin/https/jar/webserv-jstl.jar:/export/home/ws61/
          bin/https/jar/nova.jar"
      classpathsuffix="::/IS_CLASSPATH_BEGIN_DELIM:
                       //usr/share/lib/xalan.jar:
                       //export/SUNWam/lib/xmlsec.jar:
                       //usr/share/lib/xercesImpl.jar:
                       //usr/share/lib/sax.jar:
                       //usr/share/lib/dom.jar:
                       //export/SUNWam/lib/dom4j.jar:
                       //export/SUNWam/lib/jakarta-log4j-1.2.6.jar:
                       //usr/share/lib/jaxm-api.jar:
                       //usr/share/lib/saaj-api.jar:
      ```

```
                    //usr/share/lib/jaxrpc-api.jar:
                    //usr/share/lib/jaxrpc-impl.jar:
                    //export/SUNWam/lib/jaxm-runtime.jar:
                    //usr/share/lib/saaj-impl.jar:/export/SUNWam
                    //lib:/export/SUNWam/locale:
                    //usr/share/lib/mps/jss3.jar:
                    //export/SUNWam/lib/    am_sdk.jar:
                    //export/SUNWam/lib/am_services.jar:
                    //export/SUNWam/lib/am_sso_provider.jar:
                    //export/SUNWam/lib/swec.jar:
                    //export/SUNWam/lib/acmecrypt.jar:
                    //export/SUNWam/lib/iaik_ssl.jar:
                    //usr/share/lib/jaxp-api.jar:
                    //usr/share/lib/mail.jar:
                    //usr/share/lib/activation.jar:
                    //export/SUNWam/lib/servlet.jar:
                    //export/SUNWam/lib/am_logging.jar:
                    //usr/share/lib/commons-logging.jar:
                    //IS_CLASSPATH_END_DELIM:"
    envclasspathignored="true" debug="false"
    debugoptions="-Xdebug -Xrunjdwp:
    transport=dt_socket,
    server=y,suspend=n"
    javacoptions="-g"
    dynamicreloadinterval="2">
```

5 **Use the SSO samples installed on the remote SDK server for configuration purposes.**

   a. **Change to the** / *remote_SDK_server* `/SUNWam/samples/sso` **directory.**

   b. **Run** `gmake`**.**

   c. **Copy the generated class files from** / *remote_SDK_server* `/SUNWam/samples/sso` **to** / *remote_SDK_server* `/SUNWam/lib/`**.**

6 **Copy the encryption value of** `am.encryption.pwd` **from the** `AMConfig.properties` **file installed with Access Manager to the** `AMConfig.properties` **file on the remote server to which the SDK was installed.**

   The value of `am.encryption.pwd` is used for encrypting and decrypting passwords.

7 **Login into Access Manager as** `amadmin`**.**

   `http://`*AcceessManager-HostName*`:3000/amconsole`

8. **Execute the servlet by entering** `http://`*remote_SDK_host*`:58080/servlet/SSOTokenSampleServlet` **into the browser location field and validating the** `SSOToken`**.**

   `SSOTokenSampleServlet` is used for validating a session token and adding a listener. Executing the servlet will print out the following message:

   ```
   SSOToken host name: 192.18.149.33 SSOToken Principal name:
   uid=amAdmin,ou=People,dc=red,dc=iplanet,dc=com Authentication type used: LDAP
   IPAddress of the host: 192.18.149.33 The token id is
   AQIC5wM2LY4SfcyURnObg7vEgdkb+32T43+RZN30Req/BGE= Property: Company is - Sun
   Microsystems Property: Country is - USA SSO Token Validation test Succeeded
   ```

9. **Set the property** `com.iplanet.am.notification.url=` **in** `AMConfig.properties` **of the machine where the Client SDK is installed:**

   ```
   com.iplanet.am.notification.url=http://clientSDK_host.domain:port
   /servlet
       com.iplanet.services.comm.client.PLLNotificationServlet
   ```

10. **Restart the Web Server.**

11. **Login into Access Manager as** `amadmin`**.**

    `http://`*AcceessManager-HostName*`:3000/amconsole`

12. **Execute the servlet by entering** `http://`*remote_SDK_host*`:58080/servlet/SSOTokenSampleServlet` **into the browser location field and validating the** `SSOToken` **again.**

    When the machine on which the remote SDK is running receives the notification, it will call the respective listener when the session state is changed. Note that the notifications can be received only if the remote SDK is installed on a web container.

◆ ◆ ◆ CHAPTER 10

Updating and Redeploying Access Manager WAR Files

Access Manager contains a number of web archive (WAR) files. These packages contain Java servlets and JavaServer Pages™ (JSP) pages you can modify to customize Access Manager to meet your needs. The chapter contains the following sections:

WAR Files in J2EE Software Development

Access Manager is built upon the Java 2 Platform, Enterprise Edition (J2EE) platform which uses a component model to create full-scale applications. A component is self-contained functional software code assembled with other components into a J2EE application. The J2EE application components can be deployed separately on different servers. J2EE application components include the following:

- Client components such as including dynamic web pages, applets, and a Web browser that run on the client machine.
- Web components such as servlets and Java Server Pages (JSPs) that run within a web container.
- Business components, which can be code that meets the needs of a particular enterprise domain such as banking, retail, or finance. Such business components also run within the web container.
- Enterprise infrastructure software that runs on legacy machines.

Web Components

When a web browser executes a J2EE application, it deploys server-side objects known as web components. Java Server Pages (JSPs) and corresponding servlets are two such web components.

Servlets	Small Java programs that dynamically process requests and construct responses from a web browser. Servlets run within web containers.
Java Server Pages (JSPs)	Text-based documents that contain static template data such as HTML, Scalable Vector Graphics (SVG), Wireless Markup Language (WML), or eXtensible Markup Language (XML). JSPs also contain elements such as servlets that construct dynamic content.

How Web Components are Packaged

J2EE components are usually packaged separately, and then bundled together into an Enterprise Archive (EAR) file for application deployment. Web components are packaged in web application archives, also known as WAR files. Each WAR file contains servlets, JSPs, a deployment descriptor, and related resource files.

Static HTML files and JSP are stored at the top level of the WAR directory. The top-level directory contains the `WEB-INF` subdirectory which contains tag library descriptor files in addition to the following:

Server-side classes	Servlets, JavaBean components and related Java class files. These must be stored in the `WEB-INF/classes` directory.
Auxiliary JARs	Tag libraries and any utility libraries called by server-side classes. These must be stored in the `WEB-INF/lib` directory.
`web.xml`	The web component deployment descriptor is stored in the WEB-INF directory

About Access Manager WAR Files

When you customize Access Manager, sometimes you must also modify the Access Manager WAR files. The modifications in turn result in changes to the web components.

Access Manager provides two types of WAR files. One type of Access Manager WAR file is automatically built and deployed for you at installation. The `password.war` and `services.war` files are of this type. Both `password.war` and `services.war`are related to features and services

that power the Access Manager server. At installation, based on the source files in the staging directory *AccessManager-base*/web-src/, both password.war and services.war are automatically generated and deployed into the *AccessManager-base*/SUNWam/war directory. When you want to customize Access Manager features or services, you must make changes in the source files contained in the staging directory, and then regenerate and redeploy the appropriate WAR files.

Caution – When you apply a patch or an upgrade to Access Manager, any customizations you have implemented may be overwritten.

The second type of Access Manager WAR is a specialized WAR file that you must manually deploy. The amaduthdistui.war for the Distributed Authentication UI, and the amclient.war for the Client SDK are such WARs. You can install amaduthdistui.war or amclient.warthrough the JES installer, or you can manually deploy one or both of them.

The following Access Manager WAR files are located in this directory:

AccessManager-base/SUNWam/

amcommon.war	Automatically deployed at installation, and builds the Liberty IDFF profile named Identity Provider Introduction which is used in implementing a circle of trust. You do not need to redeploy this WAR.
amconsole.war	If you choose the Legacy mode option during installation, this WAR is automatically deployed at installation, and builds the legacy mode administration console. Redeploy this WAR after you make changes to *AccessManager-base*/web-src/services/console/* source files.
ampassword.war	Automatically deployed at installation, and builds the password reset feature. Redeploy this WAR after you make changes to *AccessManager-base*/web-src/password/* source files.
amserver.war	Automatically deployed at installation, and builds Access Manager service components. Redeploy this WAR after you make changes to *AccessManager-base*/web-src/services/* source files.

The following Access Manager WARs are located in this directory:

AccessManager-base/SUNWam/war.

am_server.war	Use this WAR to manually install Access Manager as a stand-alone product, and without using the JES installer. For more information, see Chapter 12, "Deploying Access Manager as a Single WAR File," in *Sun Java System Access Manager 7.1 Postinstallation Guide*.

`amclient.war`	Use this WAR to manually install the Client SDK on a container remote from the Access Manager server. For more information, see "Installing the Client SDK" on page 14.
`amauthdistui.war`	Use this WAR to manually install the Distributed Authentication UI server on a container remote from the Access Manager server. You can install this WAR using the JES installer. For more information, see Chapter 11, "Deploying a Distributed Authentication UI Server," in *Sun Java System Access Manager 7.1 Postinstallation Guide*. You can also manually deploy this WAR. For more information, see "Customizing the Distributed Authentication User Interface" on page 179.
`amconsole.war`	Access Manager uses this WAR to build the realm mode administration console. The `amconsole.war` file is automatically generated and deployed, based on the source code in *AccessManager-base*`/web-src/services/console`, when Access Manager is installed. You cannot customize this WAR.
`console.war`	Access Manager uses this WAR to build the legacy mode administration console. The `console.war` file is automatically generated and deployed, based on the source code in *AccessManager-base*`/web-src/services/console`, when Access Manager is installed. You can customize this WAR. For more information, see Chapter 11, "Customizing the Administration Console."
`introduction.war`	This WAR is related to the Liberty IDFF profile named Identity Provider Introduction which is used in implementing a circle of trust. The `introduction.war` file is automatically generated and deployed, based on the source code in *AccessManager-base*`/web-src/services/common`, when Access Manager is installed. You cannot customize this WAR.
`password.war`	Access Manager uses this WAR for the password reset service. The `password.war` file is automatically generated and deployed, based on the source code in *AccessManager-base*`/web-src/services/password`, when Access Manager is installed. You can customize this WAR. For more information, see the section "password.war" on page 145.
`services.war`	Access Manager uses this WAR to build the UI for various Access Manager Services. The `services.war` file is automatically generated and deployed, based on the source code in *AccessManager-base*`/web-src/services/services`, when Access Manager is installed. You can customize this WAR. For more information, see the section "services.war" on page 145.

password.war

The `password.war` contains files used by the Access Manager password reset service.

Files You Can Modify

You can modify the following `password.war` files:

- `web.xml` and related XML files used for constructing it are located in *AccessManager-base*`/SUNWam/web-src/password/WEB-INF/`.
- `JSP`s located in `/SUNWam/web-src/password/password/ui/`.
- Image files located in `SUNWam/web-src/password/password/images/`.
- Stylesheets located in *AccessManager-base*`/SUNWam/web-src/password/password/css/`.

Files You Must Not Modify

Do not modify the following `password.war` files. Modifying the following files may cause unintended Access Manager behaviors.

- `JAR`s located in *AccessManager-base*`/SUNWam/web-src/password/WEB-INF/lib/`.
- Tag library descriptor (`.tld`) files located in *AccessManager-base*`/web-src/password/WEB-INF/`.

services.war

The `services.war` contains files used by various Access Manager services.

Files You Can Modify

You can modify the following `services.war` files:

- `web.xml` and related XML files used for constructing it are located in *AccessManager-base*`/SUNWam/web-src/services/WEB-INF/`.
- JavaScript files are located in *AccessManager-base*`/SUNWam/web-src/services/js/`.
- JSP are located in the following directories:
  - *AccessManager-base*`/SUNWam/web-src/services/config/auth/default/`
  - *AccessManager-base*`/SUNWam/web-src/services/config/federation/default/`

  Image files are located in the following directories:

  - *AccessManager-base*`/SUNWam/web-src/services/images/`
  - *AccessManager-base*`/SUNWam/web-src/services/fed_images/`
  - *AccessManager-base*`/SUNWam/web-src/services/login_images/`

Stylesheets are located in the following directories:

- *AccessManager-base*/SUNWam/web-src/services/css/.
- *AccessManager-base*/SUNWam/web-src/services/fed_css/.

Files You Must Not Modify

Do not modify the following services.war files. Modifying the following files may cause Access Manager to fail:

- Non-modifiable JARs are located in *AccessManager-base*/SUNWam/web-src/services/WEB-INF/lib/.
- Non-modifiable Tag Library Descriptor (.tld) files are located in *AccessManager-base*/SUNWam/web-src/services/WEB-INF/.

Updating Modified WARs

Once a file within a WAR is modified, the WAR itself needs to be updated with the newly modified file. Following is the procedure to update a WAR.

▼ To Update a Modified WAR

1 Go to the directory where the WAR files are kept.

```
# cd AccessManager-base/ SUNWam/war
```

2 Run the jar command.

```
jar -uvf WARfilename.war path_to_modified_file
```

The -uvf option replaces the old file with the newly modified file. For example:

```
# jar -uvf console.war newfile/index.html
```

This command replaces the index.html file in console.war with the index.html file located in *AccessManager-base*/SUNWam/newfile.

3 Delete the modified file.

```
# rm newfile/index.html
```

Delete the modified file.

Redeploying Modified Access Manager WAR Files

Once updated, the WARs must be redeployed to their web container. The web container provides services such as request dispatching, security, concurrency, and life cycle management. The web container also gives the web components access to the J2EE APIs.

The BEA WebLogic Server 6.1 and Sun Java System Application Server web containers do not require WARs to be exploded. The servers themselves are deployed as WARs. After WAR files are installed on these servers, you must restart all related servers.

To Redeploy a WAR On BEA WebLogic Server 6.1

Run the Java command on the BEA WebLogic 6.1 Server using the following form:

```
java weblogic.deploy -url protocol://server_host:server_port
-component amconsole:WL61 _server_name
deploy WL61_admin_password deployment_URI AccessManager-base/SUNWam/WARname.war
```

where the following variables are used:

protocol://server_host:server_port	The protocol [http | https] and fully-qualified name of the Access Manager server.
WL61 _server_name	The name of the WebLogic server.
WL61_admin_password	The WebLogic administrator password.
deployment_URI	For console.war, the deployment URI is amconsole. For services.war, the deployment URI is amserver. For password.war, the deployment UIR is ampassword.
AccessManager-base	The directory where the Access Manager server is installed.
WARname.war	The name of the WAR file to deploy. [console.war | server.war | password.war]

For more complete information on the Java utility weblogic.deploy and its options, see the http://edocs.bea.com/wls/docs61/index.html.

To Redeploy a WAR File on Sun Java System Application Server 7.0

On the Application Server, run the `asadmin` command using the following form:

```
asadmin deploy -u S1AS_administrator
-w  S1AS_administrator_password -H console_server_host
-p S1AS_server_port --type web secure_flag
--contextroot deploy_uri --name deploy_uri
--instance S1AS_instanceAccessManager-base/SUNWam/WARname
```

where the following variables are used:

S1AS_administrator	Application Server administrator
S1AS_administrator_password	Application Server administrator password
console_server_host	Access Manager server host name
S1AS_server_port	Application Server port number
deploy_uri	For `console.war`, the deployment URI is `amconsole`. For `password.war`, the deployment URI is `ampassword`. For `services.war`, the deployment URI is `amservices`.
S1AS_instance/AccessManager-base	Application Server directory where Access Manager server is installed
WARname.war	The name of the WAR file to deploy. `[console.war | services.war | password.war]`

For more information on the `asadmin` deploy command and its options, see the *Sun Java System Application Server 7.0 Developer's Guide.*

Redeploying an Access Manager WAR on IBM WebSphere Application Server

For detailed instructions on how to deploy `WARs` in an IBM WebSphere Application Server container, see the documentation that comes with the product:`http://www-306.ibm.com/software/webservers/appserv/was/`

◆ ◆ ◆ CHAPTER 11

Customizing the Administration Console

The Sun Java™ System Access Manager 7.1 console is a web-based interface for creating, managing, and monitoring the identities, web services, and enforcement policies configured throughout an Access Manager deployment. It is built with Sun Java System Application Framework, a Java 2 Enterprise Edition (J2EE) framework used to help developers build functional web applications. XML files, JavaServer Pages™ (JSP) and Cascading Style Sheets (CSS) define the look of the Access Manager HTML pages.

This chapter describes the Access Manager administration console, its pluggable architecture, and how to customize the Legacy mode user interface. The chapter contains the following sections:

Note – At this time, no documentation or code samples exist for modifying the Realm mode user interface. For customized information on modifying the Realm mode user interface in your environment, contact your Sun Sales Representative.

About the Administration Console

The console is divided into three frames: Header, Navigation and Data. The Header frame displays corporate branding information as well as the first and last name of the currently logged-in user as defined in their profile. It also contains a set of tabs to allow the user to switch between the management modules, a hyperlink to the Access Manager Help system, a Search function and a Logout link. The Navigation frame on the left displays the object hierarchy of the chosen management module, and the Data frame on the right displays the attributes of the object selected in the Navigation frame.

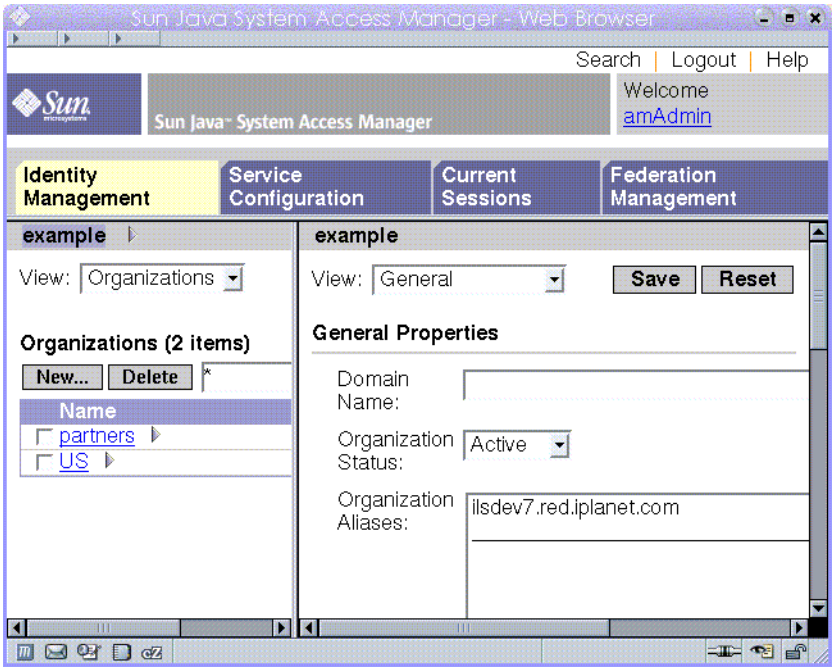

FIGURE 11–1 Legacy Mode Administration Console

For information about what the Console does and about the differences between the Realm mode and Legacy mode console interfaces, see Chapter 1, "The Access Manager Console," in *Sun Java System Access Manager 7.1 Administration Guide*.

Generating The Console Interface

When the Access Manager console receives an HTTP(S) request, it first determines whether the requesting user has been authenticated. If not, the user is redirected to the Access Manager login page supplied by the Authentication Service. After successful authentication, the user is redirected back to the console which reads all of the user's available roles, and extracts the applicable permissions and behaviors. The console is then dynamically constructed for the user based on this information. For example, users with one or more administrative roles will see the administration console view while those without any administrative roles will see the end user console view. Roles also control the actions a user can perform and the identity objects that a user sees. Pertaining to the former, the organization administrator role allows the user read and write access to all objects within that organization while a help desk administrator role only permits write access to the users' passwords. With regards to the latter, a person with a people container administrator role will only see users in the relevant people container while the

organization administrator will see all identity objects. Roles also control read and write permissions for service attributes as well as the services the user can access.

Plug-In Modules

An external application can be plugged-in to the console as a module, gaining complete control of the Navigation and Data frames for its specific functionality. In this case, a tab with the name of the custom application needs to be added to the Header frame. The application developer would create the JSPs for both left and right frames, and all view beans, and models associated with them.

Accessing the Console

The Naming Service defines URLs used to access the internal services of Access Manager. The URL used to access the Administration Console web application is:

`http://`*AcceessManager-HostName.domain_name:port/*
`amconsole`

The first time Administration Console (`amconsole`) is accessed, it brings the user to the Authentication web application (`amserver`) for authentication and authorization purposes. After login, `amserver` redirects the user to the configured success login URL. The default successful login URL is:

`http(s)://`*AcceessManager-HostName.domain_name:port/*
`amconsole/base/AMAdminFrame`

Customizing The Console

The Access Manager Legacy mode console uses JSP and CSS to define the look and feel of the pages used to generate its frames. A majority of the content is generated dynamically—based on where, and at what, the user is looking. In that regard, the modification of the content is somewhat restricted. Within the Navigation frame, the layout of the controls (the view menu), the action buttons, and the table with current objects in each JSP can be changed. In the Data frame, the content displayed is dynamically generated based on the XML service file being accessed but the layout, colors, and fonts are controlled by the `adminstyle.css` style sheet.

The Default Console Files

An administrator can modify the console by changing tags in the JSPs and CSS's. All of these files can be found in the *AccessManager-base*`/SUNWam/web-src/services/console` directory. The files in this directory provide the default Sun Java System interface. Out of the box, it contains the following subdirectories:

- `base` contains JSP that are not service-specific.
- `css` contains the `adminstyle.css` which defines styles for the console.
- `federation` contains JSP related to the Federation Management module.
- `html` contains miscellaneous HTML files.
- `images` contains images referenced by the JSP.
- `js` contains JavaScript™ files.
- `policy` contains JSP related to the Policy Service.
- `service` contains JSP related to the Service Management module.
- `session` contains JSP related to the Current Sessions (session management) module.
- `user` contains JSP related to the Identity Management module.

Note – Console-related JSP contain HTML and custom library tags. The tags are defined in tag library descriptor files (`.tld`) found in the *AccessManager-base*`/SUNWam/web-src/WEB-INF` directory. Each custom tag corresponds to a view component in its view bean. While the tags in the JSP can be removed, new tags can not be added. For more information, see the Sun Java System Application Framework at `http://docs.sun.com/app/docs/coll/S1_appframe20_en`.

---

console.war

The `console.war` contains files used by the Access Manager administration console.

Files You Can Modify

You can modify the following `console.war` files:

- `web.xml` and related XML files used for constructing it are located in *AccessManager-base* `/SUNWam/web-src/services/WEB-INF/`.
- Modifiable JavaScript files are located in *AccessManager-base* `/SUNWam/web-src/services/console/js/`.
- Modifiable JSP are located in the following directories dependant upon the service that deploys them:
  - *AccessManager-base* `/SUNWam/web-src/services/console/auth/`
  - *AccessManager-base* `/SUNWam/web-src/services/console/federation/`
  - *AccessManager-base* `/SUNWam/web-src/services/console/policy/`
  - *AccessManager-base* `/SUNWam/web-src/services/console/service/`
  - *AccessManager-base* `/SUNWam/web-src/services/console/session/`
  - *AccessManager-base* `/SUNWam/web-src/services/console/user/`

Modifiable image files are located in *AccessManager-base* `/SUNWam/web-src/services/console/images/`.

- Modifiable stylesheets are located in *AccessManager-base*`/SUNWam/web-src/services/console/css/`.

Files You Must Not Modify

Do not modify the following `console.war` files. Modifying these files may cause unintended Access Manager behaviors.

- `JARs` are located in *AccessManager-base*`/SUNWam/web-src/services/WEB-INF/lib/`.
- Tag Library Descriptor (`.tld`) files are located in *AccessManager-base*`/SUNWam/web-src/services/WEB-INF/`.

Creating Custom Organization Files

To customize the console for use by a specific organization, the *AccessManager-base*`/SUNWam/web-src/services/console` directory should first be copied, renamed and placed on the same level as the default directory. The files in this new directory can then be modified as needed.

Note – There is no standard to follow when naming the new directory. The new name can be any arbitrarily chosen value.

For example, customized console files for the organization `dc=new_org, dc=com` might be found in the *AccessManager-base*`/SUNWam/web-src/services/custom_directory` directory.

▼ To Create Custom Organization Files

1 Change to the directory where the default templates are stored:

`cd` *AccessManager-base*`/SUNWam/web-src/services`

2 Make a new directory at that level.

The directory name can be any arbitrary value. For this example, it is named *AccessManager-base*`/SUNWam/web-src/services/custom_directory/`.

3 Copy all the JSP files from the `console` directory into the new directory.

AccessManager-base`/SUNWam/web-src/services/console` contains the default JSP for Access Manager. Ensure that any image files are also copied into the new directory.

4 **Customize the files in the new directory.**

Modify any of the files in the new directory to reflect the needs of the specific organization.

5 **Modify the `AMBase.jsp` file.**

In our example, this file is found in *AccessManager-base*`/SUNWam/web-src/services/custom_directory/base`. The line `String console = "../console";` needs to be changed to `String console = "../`*new_directory_name*`";`. The `String consoleImages` tag also needs to be changed to reflect a new image directory, if applicable. The contents of this file are copied in "Creating Custom Organization Files" on page 153.

```
<!--
 Copyright © 2002 Sun Microsystems, Inc. All rights reserved.
 Use is subject to license terms.
-->

<% String console = "../console";
   String consoleUrl = console + "/";
   String consoleImages = consoleUrl + "images";
%>
```

6 **Change the value of the JSP Directory Name attribute in the Administration Service to match that of the directory created in "Creating Custom Organization Files" on page 153.**

The JSP Directory Name attribute points the Authentication Service to the directory which contains an organization's customized console interface. Using the console itself, display the services registered to the organization for which the console changes will be displayed. If the Administration Service is not visible, it will need to be registered. For information on registering services, see the Administration Guide.

Once the new set of console files have been modified, the user would need to log into the organization where they were made in order to see any changes. Elaborating on our example, if changes are made to the JSP located in the *AccessManager-base*`/SUNWam/web-src/services/custom_directory` directory, the user would need to login to that organization using the URL:

`http://` *server_name.domain_name:port*`//`
service_deploy_uri`/UI/Login?org=`
custom_directory_organization.

Alternate Customization Procedure

The console can also be modified by simply replacing the default images in *AccessManager-base*/SUNWam/web-src/services/console/images , with new, similarly named images.

Miscellaneous Customizations

Included in this section are procedures for several specific customizations available to administrators of the Access Manager console.

To Modify The Service Configuration Display

A *service* is a group of attributes that are managed together by the Access Manager console. Out-of-the-box, Access Manager loads a number of services it uses to manage its own features. For example, the configuration parameters of the Logging Service are displayed and managed in the Access Manager console, while code implementations within Access Manager use the attribute values to run the service.

To Modify The User Profile View

The Access Manager console creates a default User Service view based on information defined in the amUser.xml service file.

A modified user profile view with functionality more appropriate to the organization's environment can be defined by creating a new ViewBean and/or a new JSP. For example, an organization might want User attributes to be formatted differently than the default vertical listing provided. Another customization option might be to break up complex attributes into smaller ones. Currently, the server names are listed in one text field as:

protocol://*Access Manager_host*. *domain*:*port*

Instead, the display can be customized with three text fields:

protocol_chooser_field://*server_host_field* :*port_number_field*

A third customization option might be to add JavaScript to the ViewBean to dynamically update attribute values based on other defined input. The custom JSP would be placed in the following directory: *AccessManager-base*/SUNWam/web-src/services/console/user. The ViewBean is placed in the classpath com.iplanet.am.console.user. The value of the attribute User Profile Display Class in the Administration Service (iplanet-am-admin-console-user-profile-class in the amAdminConsole.xml service file) would then be changed to the name of the newly created ViewBean. The default value of this attribute is com.iplanet.am.console.user.UMUserProfileViewBean.

Display Options For The User Profile Page

There are a number of attributes in the Administration Service that can be selected to display certain objects on the User Profile page. Display User's Roles, Display User's Groups and User Profile Display Options specify whether to display the roles assigned to a user, the groups to which a user is a member and the schema attributes, respectively. More information on these service attributes can be found in the Administration Guide.

To Localize The Console

All textual resource strings used in the console interface can be found in the `amAdminModuleMsgs.properties` file, located in *AccessManager-base*`/SUNWam/locale/`. The default language is English (`en_US`). Modifying this file with messages in a foreign language will localize the console.

To Display Service Attributes

Service attributes are defined in XML service files based on the `sms.dtd` . In order for a particular service attribute to be displayed in the console, it must be configured with the `any` XML attribute. The any attribute specifies whether the service attribute for which it is defined will display in the Access Manager console.

To Customize Interface Colors

All the colors of the console are configurable using the Access Manager style sheet `adminstyle.css` located in the *AccessManager-base*`/SUNWam/web-src/services/console/css` directory. For instance, to change the background color for the navigation frame, modify the `BODY.navFrame` tag; or to change the background color for the data frame, modify the `BODY.dataFrame`. The tags take either a text value for standard colors (blue, green, red, yellow, etc.) or a hexadecimal value (#ff0000, #aadd22, etc.). Replacing the default with another value will change the background color of the respective frame after the page is reloaded in the browser. "Miscellaneous Customizations" on page 155 details the tag in `adminstyle.css`.

EXAMPLE 11–1 `BODY.navFrame` Portion of `adminstyle.css`

```
BODY.navFrame   {
                color: black;
                background: #ffffff;
                }
```

To Change The Default Attribute Display Elements

The console auto-generates Data frame pages based on the definition of a service's attributes in an XML service definition file. Each service attribute is defined with the XML attributes `type`, `uitype` and `syntax`. `Type` specifies the kind of value the attribute will take. `uitype` specifies the HTML element displayed by the console. `syntax` defines the format of the value. The values of these attributes can be mixed and matched to alter the HTML element used by the console to display the values of the attributes. For example, by default, an attribute of the `single_choice` type displays its choices as a drop down list in which only one choice can be selected. This list can also be presented as a set of radio buttons if the value of the `uitype` attribute is changed to `radio`. "Miscellaneous Customizations" on page 155 illustrates this concept.

EXAMPLE 11–2 `uitype` XML Attribute Sample

```
<AttributeSchema name="test-attribute"
                   type="single_choice"
                   syntax="string"
                   any="display"
                   uitype="radio"
                   i18nKey="d105">
                   <ChoiceValues>
<ChoiceValue i18nKey="u200">Daily</ChoiceValue>
<ChoiceValue i18nKey="u201">Weekly</ChoiceValue>
<ChoiceValue i18nKey="u202">Monthly</ChoiceValue>
                   </ChoiceValues>
                   <DefaultValues>
                       <Value>Daily</Value>
                   </DefaultValues>
               </AttributeSchema>
```

"Miscellaneous Customizations" on page 155 is a listing of the possible values for each attribute, and the corresponding HTML element that each will display based on the different groupings.

TABLE 11–1 Service Attribute Values and Corresponding Display Elements

type Value	syntax Value	uitype Value	Element Displayed In Console
single_choice	string	No value defined	pull-down menu choices
		radio	radio button choices
Single	boolean	No value defined	checkbox
		radio	radio button

TABLE 11–1 Service Attribute Values and Corresponding Display Elements *(Continued)*

type Value	syntax Value	uitype Value	Element Displayed In Console
	string	No value defined	text field
		link	hyperlink
		button	clickable button
	password	No value defined	text field
	paragraph	No value defined	scrolling text field
list	string	No value defined	Add/Delete name list
		`name_value_list`	Add/Edit/Delete name list
`multiple_choice`	string	No value defined	choice list

To Add A Module Tab

The section "Plug-In Modules" mentions the capability to plug-in external applications as modules. Once this is accomplished, the module needs to be accessible via the console by adding a new module tab. Label information for module tabs are found in the `amAdminModuleMsgs.properties` console properties file located in *AccessManager-base*`/SUNWam/locale/`. To add label information for a new module, add a key and value pair similar to `module105_NewTab=My New Tab`. "Miscellaneous Customizations" on page 155 illustrates the default pairs in the file.

EXAMPLE 11–3 Module Tab Key And Value Pairs

```
module101_identity=Identity Management
module102_service=Service Configuration
module103_session=Current Sessions
module104_federation=Federation Management
```

The module name and a URL for the external application also need to be added to the View Menu Entries attribute in the Administration Service (or `iplanet-am-admin-console-view-menu` in the `amAdminConsole.xml` service file). When a module tab in the Header frame is clicked, this defined URL is displayed in the Navigation frame. For example, to define the display information for the tab sample, an entry similar to `module105_NewTab|/amconsole/`*custom_directory* */custom_NavPage* would be added to the View Menu Entries attribute in the Administration Service.

Note – The console retrieves all the entries from this attribute and sorts them by i18n key. This determines the tab display order in the Header frame.

After making these changes and restarting Access Manager, a new tab will be displayed with the name My New Tab.

To Display Container Objects

In order to create and manage LDAP organizational units (referred to as *containers* in the console), the following attributes need to be enabled (separately or together) in the Administration Service.

- Display Containers In Menu—Containers are organizational units as viewed using the Access Manager console. If this option is selected, the menu choice Containers will be displayed in the View menu for top-level Organizations, Sub-Organizations and other containers.
- Show People Containers—People containers are organizational units containing user profiles. If this option is selected, the menu choice People Containers will be displayed in the View menu for Organizations, Containers and Sub-Organizations.
- Show Group Containers—Group containers are organizational units containing groups. If this option is selected, the menu choice Group Containers will be displayed in the View menu for Organizations, Containers and Group Containers.

Viewing any of these display options is also dependent on whether the Enable User Management attribute is selected in the Administration Service. (This attribute is enabled by default after a new installation.) More information on these attributes can be found in the Administration Guide.

Console APIs

The public console API package is named `com.iplanet.am.console.base.model` . It contains interfaces that can be used to monitor and react to events that occur in the console. This *listener* can be called when the user executes an action on the console that causes an event. An event can have multiple listeners registered on it. Conversely, a listener can register with multiple events. Events that might be used to trigger a listener include:

- Displaying a tab in the Header frame.
- Creating or deleting identity-related objects.
- Modifying the properties of an identity-related object.
- Sending attribute values to the console ViewBean for display purposes.

When a listener is created all the methods of that interface must be implemented thus, the methods in the `AMConsoleListener` interface must be implemented. The `AMConsoleListenerAdapter` class provides default implementations of those methods and can be used instead. Creating a console event listener includes the following:

▼ To Create a Console Event Listener

1. **Write a console event listener class or implement the default methods in the `AMConsoleListenerAdapter` class.**
2. **Compile the code.**
3. **Register the listener in the Administration Service.**

   Access Manager includes a sample implementation of the `ConsoleEventListener`. The *Sun Java System Access Manager 7.1 Java API Reference* also contains more detailed information on the listener interfaces and class.

Precompiling the Console JSP

Each JSP is compiled when it is first accessed. Because of this, there is a delay when displaying the HTML page on the browser. To avoid this delay, the system administrator can precompile the JSP by running the following command:

WebServer_install_directory`/servers/bin/https/bin/jspc -webapp`
AccessManager-base`/SUNWam/web-src/services`

where, by default, *WebServer_install_directory* is `/opt/SUNWwbsvr`.

Console Samples

Sample files have been included to help understand how the Access Manager console can be customized. The samples include instructions on how to:

Modify User Profile Page

This sample modifies the user interface by adding a hyperlink that allows an existing user to change their configured password. It is in the `ChangeUserPassword` directory.

Create A Tabbed Identity Management Display

This sample creates a custom user profile which displays the profile with three tabs. The sample is in the `UserProfile` directory.

ConsoleEventListener

This sample displays the parameters passed to `AMConsoleListener` class in the `amConsole` debug file. It is in the `ConsoleEventListener` directory.

Add Administrative Function

This sample adds functionality to the Identity Management module that allows an administrator to move a user from one organization to other. It is in the `MoveUser` directory.

Add A New Module Tab

This sample adds a new tab into the Header frame. This tab will connect to an external application and can be configured using the console. It is in the `NewTab` directory.

Create A Custom User Profile View

This sample creates a custom user profile view to replace the default user profile view. A different user profile view can be created for each configured organization. A custom class would need to be written that extends the default user profile view bean. This class would then be registered in the User Profile Display Class attribute of the Administration Service. There is an example of how to do this in the samples directory. This sample is in the `UserProfile` directory.

These samples are located in *AccessManager-base*`/SUNWam/samples/console`. Open the `README` file in this directory for general instructions. Each specific sample directory also contains a `README` file with instructions relevant to that sample.

Note – The console samples are only available when Access Manager is installed on the Solaris™ operating system.

◆ ◆ ◆ CHAPTER 12

Customizing the Authentication User Interface

The Authentication Service provides the web-based Graphical User Interface (GUI) for all default and custom authentication modules installed in the Sun Java™ System Access Manager 7.1 deployment. This interface provides a dynamic and customizable means for gathering authentication credentials by presenting the web-based login requirement pages to a user requesting access.

The Authentication Service GUI is built on top of JATO (J2EE Assisted Take-Off), a Java 2 Enterprise Edition (J2EE) presentation application framework. This framework is used to help developers build complete functional Web applications. You can customize this user interface per client type, realm, locale, or service.

For more information about what the Authentication Service does and how it works, see Chapter 3, "Authentication," in *Sun Java System Access Manager 7.1 Technical Overview* and Chapter 11, "Deploying a Distributed Authentication UI Server," in *Sun Java System Access Manager 7.1 Postinstallation Guide*.

The following topics are covered in this chapter:

User Interface Files You Can Modify

The authentication GUI dynamically displays the required credentials information depending upon the authentication module invoked at run time. The "User Interface Files You Can Modify" on page 163 lists the types of files you can modify to convey custom representations of Login pages, Logout pages, and error messages. Detailed information is provided in following sections.

TABLE 12–1 Authentication User Interface Files and Their Locations at Installation

File Type	Default Location
"Staging Area for Files to be Customized" on page 164	*AccessManager-base*/SUNWam/web-src/services
"Java Server Pages" on page 165	*AccessManager-base*/SUNWam/web-src/services/config/auth/default
"XML Files" on page 167	*AccessManager-base*/SUNWam/web-src/services/config/auth/default
"JavaScript Files" on page 170	*AccessManager-base*/SUNWam/web-src/services/js
"Cascading Style Sheets" on page 171	<*AccessManager-base* /SUNWam/web-src/services/css
"Images" on page 171	*AccessManager-base*/SUNWam/web-src/services/login_images
"Localization Files" on page 172	*AccessManager-base*/SUNWam/locale

To access the default Login page, use the following URL:

```
<server_protocol>://<server_host>.<server_domain>:<server_port>/
            <service_deploy_uri>/UI/Login
```

To access the default Logout page, use the following URL:

```
<server_protocol>://<server_host>.<server_domain>:<server_port>/
            <service_deploy_uri>/UI/Logout
```

Staging Area for Files to be Customized

When Access Manager is installed, a staging area exists in the following location:

AccessManager-base/SUNWam/web-src/services

This directory content is identical to the content of the services.war.

The *AccessManager-base*/SUNWam/web-src/services contains all the files you need to modify the authentication GUI. When you install Access Manager on Sun Java System Application Server, on Sun Java System Web Server, or on BEA WebLogic Web Server, services.war (the services web application) is automatically installed and deployed.

If you install Access Manager on other web containers, you may have to manually deploy services.war. See the documentation that comes with the web container.

Once you've modified the authentication GUI files in the staging area, in order to see the changes in the actual GUI, you must update and then redeploy services.war. See "Updating and Redeploying services.war" on page 177.

Java Server Pages

All authentication GUI pages are `.jsp` files with embedded JATO tags. You do not need to understand JATO to customize Access Manager GUI pages. Java server pages handle both the UI elements and the disciplines displayed through peer ViewBeans. By default, JSP pages are installed in the following directory:
AccessManager-base`/SUNWam/web-src/services/config/auth/default`

Java server pages are looked up from the deployed location. In previous Access Manager versions, the Java server pages were looked up from the installed location.

Customizing the Login Page

The Login page is a common Login page used by most authentication modules except for the Membership module. For all other modules, at run time the Login page dynamically displays all necessary GUI elements for the required credentials. For example, the LDAP authentication module Login page dynamically displays the LDAP module header, LDAP User name, and Password fields.

You can customize the following Login page UI elements:

- Module Header text
- User Name label and field
- Password label and field
- Choice value label and field.

  The field is a radio button by default, but can be change to a check box.
- Image (at the module level)
- Login button

Customizing JSP Templates

Use the JSP templates to customize the look and feel presented in the graphical user interface (GUI). "Customizing JSP Templates" on page 165 provides descriptions of templates you can customize. The templates are located in the following directory:

AccessManager-base`/SUNWam/web-src/services/config/auth/default`

TABLE 12–2 Customizable JSP Templates

File Name	Purpose
`account_expired.jsp`	Informs the user that their account has expired and should contact the system administrator.

TABLE 12–2 Customizable JSP Templates *(Continued)*

File Name	Purpose
auth_error_template.jsp	Informs the user when an internal authentication error has occurred. This usually indicates an authentication service configuration issue.
authException.jsp	Informs the user that an error has occurred during authentication.
configuration.jsp	Configuration error page that displays during the Self-Registration process.
disclaimer.jsp	This is a customizable disclaimer page used in the Self-registration authentication module.
Exception.jsp	Informs the user that an error has occurred.
invalidAuthlevel.jsp	Informs the user that the authentication level invoked was invalid.
invalid_domain.jsp	Informs the user that no such domain exists.
invalidPassword.jsp	Informs the user that the password entered does not contain enough characters.
invalidPCookieUserid.jsp	Informs the user that a persistent cookie user name does not exist in the persistent cookie domain.
Login.jsp	This is a Login/Password template.
login_denied.jsp	Informs the user that no profile has been found in this domain.
login_failed_template.jsp	Informs the user that authentication has failed.
Logout.jsp	Informs the user that they have logged out.
maxSessions.jsp	Informs the user that the maximum sessions have been reached.
membership.jsp	A login page for the Self-registration module.
Message.jsp	A generic message template for a general error not defined in one of the other error message pages.
missingReqField.jsp	Informs the user that a required field has not been completed.
module_denied.jsp	Informs the user that the user does not have access to the module.
module_template.jsp	A customizable module page.
new_org.jsp	This page is displayed when a user with a valid session in one organization wants to login to another organization.
noConfig.jsp	Informs the user that no module configuration has been defined.
noConfirmation.jsp	Informs the user that the password confirmation field has not been entered.
noPassword.jsp	Informs the user that no password has been entered.

TABLE 12–2 Customizable JSP Templates *(Continued)*

File Name	Purpose
`noUserName.jsp`	Informs the user that no user name has been entered. It links back to the login page.
`noUserProfile.jsp`	Informs the user that no profile has been found. It gives them the option to try again or select New User and links back to the login page.
`org_inactive.jsp`	Informs the user that the organization they are attempting to authenticate to is no longer active.
`passwordMismatch.jsp`	This page is called when the password and confirming password do not match.
`profileException.jsp`	Informs the user that an error has occurred while storing the user profile.
`Redirect.jsp`	This page carries a link to a page that has been moved.
`register.jsp`	A user self-registration page.
`session_timeout.jsp`	Informs the user that their current login session has timed out.
`userDenied.jsp`	Informs the user that they do not possess the necessary role (for role-based authentication.)
`userExists.jsp`	This page is called if a new user is registering with a user name that already exists.
`user_inactive.jsp`	Informs the user that they are not active.
`userPasswordSame.jsp`	Called if a new user is registering with a user name field and password field have the same value.
`wrongPassword.jsp`	Informs the user that the password entered is invalid.

XML Files

XML files describe the authentication module-specific properties based on the Authentication Module Properties DTD file: *AccessManager-base*`/SUNWam/Auth_Module_Properties.dtd`. Access Manager defines required credentials and callback information for each of the default authentication modules. By default, Authentication XML files are installed in the following directory:

AccessManager-base`/SUNWam/web-src/services/config/auth/default` The table "XML Files" on page 167 provides descriptions of the authentication module configuration files.

XML files are looked up from the deployed location. In previous Access Manager versions, the XML files were looked up from the installed location.

TABLE 12–3 List of Authentication Module Configuration Files

File Name	Purpose
AD.xml	Defines a Login screen for use with Active Directory authentication.
`Anonymous.xml`	For anonymous authentication, although there are no specific credentials required to authenticate.
`Application.xml`	Needed for application authentication.
`Cert.xml`	For certificate-based authentication although there are no specific credentials required to authenticate.
`HTTPBasic.xml`	Defines one screen with a header only as credentials are requested via the user's web browser.
`JDBC.xml`	Defines a Login screen for use with Java Database Connectivity (JDBC) authentication.
`LDAP.xml`	Defines a Login screen, a Change Password screen and two error message screens (Reset Password and User Inactive).
`Membership.xml`	Default data interface which can be used to customize for any domain.
`MSISDN.xml`	Defines a Login screen for use with Mobile Subscriber ISDN (MSISDN).
`NT.xml`	Defines a Login screen.
`RADIUS.xml`	Defines a Login screen and a RADIUS Password Challenge screen.
`SafeWord.xml`	Defines two Login screens: one for User Name and the next for Password.
`SAML.xml`	Defines a Logins screen for Security Assertion Markup Language (SAML) authentication.
`SecurID.xml`	Defines five Login screens including `UserID` and `Passcode`, PIN mode, and Token Passcode.
`Unix.xml`	Defines a Login screen and an Expired Password screen.

Callbacks Element

The Callbacks element is used to define the information a module needs to gather from the client requesting authentication. Each Callbacks element signifies a separate screen that can be called during the authentication process.

Nested Elements

The following table describes nested elements for the `Callbacks` element.

Element	Required	Description
`NameCallback`	*	Requests data from the user; for example, a user identification.
`PasswordCallback`	*	Requests password data to be entered by the user.
`ChoiceCallback`	*	Used when the application user must choose from multiple values.
`ConfirmationCallback`	*	Sends button information such as text which needs to be rendered on the module's screen to the authentication interface.
`HttpCallback`	*	Used by the authentication module with HTTP-based handshaking negotiation.
`SAMLCallback`		Used for passing either Web artifact or SAML POST response from SAML service to the SAML authentication module when this module requests for the respective credentials. This authentication module behaves as SAML recipient for both (Web artifact or SAML POST response) and retrieves and validates SAML assertions.

Attributes

The following table describes attributes for the `Callbacks` element.

`length`	The number or length of callbacks.
`order`	Is the sequence of the group of callbacks.
`timeout`	Number of seconds the user has to enter credentials before the page times out. Default is 60.
`template`	Defines the UI `.jsp` template name to be displayed.
`image`	Defines the UI or page-level image attributes for the UI customization
`header`	Text header information to be displayed on the UI. Default is Authentication.

`error`	Indicates whether authentication framework/module needs to terminate the authentication process. If yes, then the value is true. Default is `false` .

ConfirmationCallback Element

The `ConfirmtationCallback` element is used by the authentication module to send button information for multiple buttons. An example is the button text which must be rendered on the UI page. The `ConfirmationCallback` element also receives the selected button information from the UI.

Nested Element

`ConfirmationCallback` has one nested element named `OptionValues`. The `OptionValues` element provides a list or an array of button text information to be rendered on the UI page.`OptionValues` takes no attributes.

If there is only one button on the UI page, then the module is not required to send this callback. If `ConfirmationCallback` is not provided through the Authentication Module properties XML file, then `anAuthUI.properties` will be used to pick and display the button text or label for the Login button. `anAuthUI.properties` is the global UI i18n properties file for all modules.

Callbacks length value should be adjusted accordingly after addition of the new callback.

Example:

```
<ConfirmationCallback>
    <OptionValues>
        <OptionValue>
            <Value> <required button text> </Value>
        </OptionValue>
    </OptionValues>
</ConfirmationCallback>
```

JavaScript Files

JavaScript files are parsed within the `Login.jsp` file. You can add custom functions to the JavaScript files in the following directory: *AccessManager-base*`/SUNWam/web-src/services/js` .

The Authentication Service uses the following JavaScript files:

`auth.js`	Used by `Login.jsp` for parsing all module files to display login requirement screens.
`browserVersion.js`	Used by `Login.jsp` to detect the client type.

Cascading Style Sheets

To define the look and feel of the UI, modify the cascading style sheets (CSS) files. Characteristics such as fonts and font weights, background colors, and link colors are specified in the CSS files. You must choose the appropriate `.css` file for your browser in order to customize the look and feel on the User Interface.

In the appropriate `.css` file, change the background-color attribute. Examples:

```
.button-content-enabled { background-color:red; }
button-link:link, a.button-link:visited { color: #000;
background-color: red;
text-decoration: none; }
```

A number of browser-based CSS files are installed with Access Manager in the following directory:

AccessManager-base`/SUNWam/web-src/services/css`.

The following table provides a brief description of each CSS file.

TABLE 12–4 Cascading Style Sheets

File Name	Purpose
`css_generic.css`	Configured for generic web browsers.
`css_ie5win.css`	Configured specifically for Microsoft® Internet Explorer v.5 for Windows®.
`css_ns4sol.css`	Configured specifically for Netscape™ Communicator v. 4 for Solaris™.
`css_ns4win.css`	Configured specifically for Netscape Communicator v.4 for Windows.
`styles.css`	Used in JSP pages as a default style sheet.

Images

The default authentication GUI is branded with Sun Microsystems, Inc. logos and images. By default, the GIF files are installed in the following directory:

`SUNWam/web-src/services/login_images`

These images can be replaced with images relevant to your company. The following table provides a brief description for each GIF image used for the default GUI.

TABLE 12–5 Sun Microsystems Branded GIF Images

File Name	Purpose
`Identity_LogIn.gif`	Sun Java System Access Manager banner across the top.
`Registry_Login.gif`	No longer used.
`bannerTxt_registryServer.gif`	No longer used.
`logo_sun.gif`	Sun Microsystems logo in the upper right corner.
`spacer.gif`	A one pixel clear image used for layout purposes.
`sunOne.gif`	Sun Java System logo in the lower right corner.

Localization Files

Localization files are located in the following directory: *AccessManager-base*`/SUNWam/locale`

These are `i18n` properties files global to the Access Manager instance. A localization properties file, also referred to as an *i18n (internationalization) properties file* specifies the screen text and error messages that an administrator or user will see when directed to an authentication module's attribute configuration page. Each authentication module has its own properties file that follows the naming format `amAuth`*modulename*`.properties` ; for example, `amAuthLDAP.properties`. They are located in *AccessManager-base*`/SUNWam/locale/`. The default character set is ISO-8859-1 so all values are in English, but Java applications can be adapted to various languages without code changes by translating the values in the localization properties file.

The following table summarizes the localization properties files configured for each module. These files can be found in *AccessManager-base*`/SUNWam/locale`.

TABLE 12–6 List of Localization Properties Files

File Name	Purpose
`amAuth.properties`	Defines the parent Core Authentication Service.
`amAuthAD.properties`	Defines the Active Directory Authentication Module.
`amAuthAnonymous.properties`	Defines the Anonymous Authentication Module.
`amAuthApplication.properties`	For Access Manager internal use only. Do not remove or modify this file.
`amAuthCert.properties`	Defines the Certificate Authentication Module.
`amAuthConfig.properties`	Defines the Authentication Configuration Module.

TABLE 12–6 List of Localization Properties Files *(Continued)*

File Name	Purpose
`amAuthContext.properties`	Defines the localized error messages for the `AuthContext` Java class.
`amAuthContextLocal.properties`	For Access Manager internal use only. Do not remove or modify this file.
`amAuthHTTPBasic.properties`	Defines the HTTP Basic Authentication Module.
`amAuthJDBC.properties`	Defines the Java Database Connectivity (JDBC) Authentication Module.
`amAuthLDAP.properties`	Defines the LDAP Authentication Module.
`amAuthMembership.properties`	Defines the Membership Authentication Module.
`amAuthMSISDN.properties`	Defines the Mobile Subscriber ISDN Authentication Module.
`amAuthNT.properties`	Defines the Windows NT Authentication Module.
`amAuthRadius.properties`	Defines the RADIUS Authentication Module.
`amAuthSafeWord.properties`	Defines the Safeword Authentication Module.
`amAuthSAML.properties`	Defines the Security Assertion Markup Language (SAML) Authentication Module.
`amAuthSecurID.properties`	Defines the SecurID Authentication Module.
`amAuthUI.properties`	Defines labels used in the authentication user interface.
`amAuthUnix.properties`	Defines the UNIX Authentication Module.

Customizing Branding and Functionality

You can modify JSP templates and module configuration properties files to reflect branding or functionality specified for any of the following:

- Organization of the request
- SubOrganization of the request.
- Locale of the request
- Client Path
- Client Type information of the request
- Service Name (serviceName)

▼ To Modify Branding and Functionality

1 Go to the directory where default JSP templates are stored.

`cd` *AccessManager-base*`/SUNWam/web-src/services/config/auth`

2 Create a new directory.

Use the appropriate customized directory path based on the level of customization. Use the following forms:

```
org_locale/orgPath/filePath
     org/orgPath/filePath
     default_locale/orgPath/filePath
     default/orgPath/filePath
```

In these examples,

`orgPath` represents `subOrg1/subOrg2`

`filePath` represents `clientPath + serviceName`

`clientPath` represents `clientType/sub-clientType`

In these paths, SubOrg, Locale, Client Path, Service Name (which represents `orgPath` and `filePath`) are optional. The organization name you specify may match the organization attribute set in the Directory Server. For example, if the organization attribute value is `SunMicrosystems`, then the organization customized directory should also be `SunMicrosystems`. If no organization attribute exists, then use the lowercase value of the organization name (`sunmicrosystems`).

For example, for the following attributes:

`org = SunMicrosystems`

`locale = en`

`subOrg = solaris`

`clientPath = html/` *customerName*`/`

`serviceName = paycheck`

customized directory paths would be:

`SunMicrosystems_en/solaris/html/` *customerName* `/paycheck`

`SunMicrosystems/solaris/html/` *customerName* `/paycheck`

`default_en/solaris/html/` *customerName*`/paycheck`

`default/solaris/html/` *customerName* `/paycheck`

3. **Copy the default templates.**

   Copy all the JSP templates (`*.jsp`) and authentication module configuration properties `XML` files (`*.xml`) from the default directory:

   AccessManager-base `/SUNWam/web-src/services/config/auth/default`

   to the new directory:

   AccessManager-base `/SUNWam/web-src/services/config/ auth/`*CustomizedDirectoryPath*

4. **Customize the files in the new directory.**

   The files in the new directory can be customized if necessary, but not this is not required. See "Customizing the Login Page" on page 165 and "Customizing JSP Templates" on page 165 for information on what you can modify.

5. **Update and redeploy `services.war`.**

   Once you've modified the authentication GUI files, in order to see the changes in the actual GUI, you must update and then redeploy `services.war`. See "Updating and Redeploying services.war" on page 177 in this chapter for instructions. See Chapter 10, "Updating and Redeploying Access Manager WAR Files," for general information on updating and redeploying Access Manager `.war` files.

6. **Restart both Access Manager and the web container server.**

Customizing the Self-Registration Page

You can customize the Self-registration page which is part of Membership authentication module. The default data and interface provided with the Membership authentication module is generic and can work with any domain. You can configure it to reflect custom data and information. You can add custom user profile data or fields to register or to create a new user.

▼ To Modify the Self-Registration Page

1. **Customize the `Membership.xml` file.**

   By default, the first three data fields are required in the default Membership Module configuration:

   - User name
   - User Password
   - Confirm User Password

     You can specify which data is requested, which is required, and which is optional. The sample below illustrates how to add a telephone number as requested data.

You can specify or add data which should be requested from a user as part of the User Profile. By default you can specify or add any attributes from the following `objectClasses`:

- `top`
- `person`
- `organizationalPerson`
- `inetOrgPerson`
- `iplanet-am-user-service`
- `inetuser`

Administrators can add their own user attributes to the User Profile.

2 Update and redeploy `services.war`.

Once you've modified the authentication GUI files, in order to see the changes in the actual GUI, you must update and then redeploy `services.war`. See "Updating and Redeploying services.war" on page 177 in this chapter for instructions. See Chapter 10, "Updating and Redeploying Access Manager WAR Files," for general information on updating and redeploying Access Manager `.war` files.

3 Restart both Access Manager and the web container server.

```
<Callbacks length="9" order="16" timeout="300"
header="Self Registration" template="register.jsp" >

    <NameCallback isRequired="true" attribute="uid" >
    <Prompt> User Name: </Prompt>
    </NameCallback>

    <PasswordCallback echoPassword="false" isRequired="true"
                attribute="userPassword" >
    <Prompt> Password: </Prompt>
    </PasswordCallback>

    <PasswordCallback echoPassword="false" isRequired="true" >
    <Prompt> Confirm Password: </Prompt>
    </PasswordCallback>

    <NameCallback isRequired="true" attribute="givenname" >
    <Prompt> First Name: </Prompt>
    </NameCallback>

    <NameCallback isRequired="true" attribute="sn" >
    <Prompt> Last Name: </Prompt>
    </NameCallback>

    <NameCallback isRequired="true" attribute="cn" >
    <Prompt> Full Name: </Prompt>
```

```
    </NameCallback>

    <NameCallback attribute="mail" >
    <Prompt> Email Address: </Prompt>
    </NameCallback>


<NameCallback isRequired="true"attribute="telphonenumber">
<Prompt> Tel:</Prompt>
</NameCallback>

    <ConfirmationCallback>

        <OptionValues>
        <OptionValue>
        <Value> Register </Value>
        </OptionValue>
        <OptionValue>
        <Value> Cancel </Value>
        </OptionValue>
        </OptionValues>

    </ConfirmationCallback>

</Callbacks>
```

Updating and Redeploying services.war

If Access Manager is installed on BEA WebLogic, IBM WebSphere, or Sun ONE Application Server, you must update and redeploy `services.war` before you can see any changes in the user interface. Once you've made changes to the authentication GUI files, regardless of the brand of web container you're using, it is a good practice to update and redeploy the `services.war` file. When you update and redeploy `services.war`, you overwrite the default GUI files with your changes, and the changed files are placed in their proper locations. The section "Staging Area for Files to be Customized" on page 164 provides background information on this file.

▼ To Update services.war

1 **cd** *AccessManager-base*`/SUNWam`

This is the directory in which the `WARs` are kept.

2 **jar -uvf** *WARfilename***.war <** *path_to_modified_file***>**

The `-uvf` option replaces the old file with the newly modified file. For example:

```
jar -uvf services.war newfile/index.html
```

replaces the `index.html` file in `console.war` with the `index.html` file located in *AccessManager-base*`/SUNWam/newfile`.

3 `rm newfile/index.html`

Deletes the modified file.

To Redeploy services.war

The `services.war` will be in the following directory:

```
AccessManager-base/SUNWam
```

Depending upon the brand of web container you are using, execute one of the following commands.

On BEA WebLogic

```
java weblogic.deploy -url ServerURL -component
        {ServerDeployURI}: { WL61 Server}
        deploy WL61AdminPassword {ServerDeployURI }
```

{*AccessManager-base*}`/{SUNWam}/services.war`

In this example,

ServerURL uses the form *protocol*:// *host*:*port*
Example: `http://abc.com:58080`

ServerDeployURI represents the server Universal Resource Identifier
Example: `amserver`

WL61 Server represents the Weblogic Server nam.e
Example: *name*.com

On Sun ONE Application Server

```
asadmin deploy -u IAS7Admin -w IAS7AdminPassword -H
        HostName -p IAS7AdminPort
            --type web SECURE_FLAG --contextroot
```

ServerDeployURI `--name amserver --instance IAS7Instance`
{*AccessManager-base*}`/{SUNWam}/services.war`

On IBM WebSphere

See the deployment documentation that comes with the IBM WebSphere product:

`http://www-306.ibm.com/software/webservers/appserv/was/`

Customizing the Distributed Authentication User Interface

Access Manager provides a remote Authentication user interface component to enable secure, distributed authentication across two firewalls. You can install the remote authentication user interface component on any servlet-compliant web container within the non-secure layer of an Access Manager deployment. The remote component works with Authentication client APIs and authentication utility classes to authenticate web users. The remote component is customizable and uses a JATO presentation framework.

For detailed information on how Distributed Authentication works, see "Distributed Authentication User Interface" in *Sun Java System Access Manager 7.1 Technical Overview* and Chapter 11, "Deploying a Distributed Authentication UI Server," in *Sun Java System Access Manager 7.1 Postinstallation Guide*.

Once the Distributed Authentication component is installed and deployed, you can modify the JSP templates and module configuration properties files to reflect branding and specific functionality for any of the following:

Organization/SubOrganization	This is the organization or sub-organization of the request.
Locale	Locale of the request.
Client Path	Client Type information of the request.
Service Name (`serviceName`)	Service name for service-based authentication.

▼ To Customize the Distributed Authentication User Interface

Before You Begin The Distributed Authentication User Interface package must already be installed. For detailed installation instructions, see "Installing and Configuring a Distributed Authentication UI Server Using the Java ES Installer" in *Sun Java System Access Manager 7.1 Postinstallation Guide*.

1. **Explode the Distributed Authentication User Interface WAR.**
2. **At the command line, go to the directory where the default JSP templates are stored.**

   Example:

`cd` *DistributedAuth-base*`/config/auth`

where *DistributedAuth-base* is the directory where the Distributed Authentication User Interface package is exploded.

3 Create a new directory using the appropriate directory path based on the level of customization.

Use the following form:

```
org_locale/orgPath/filePath
        org/orgPath/filePath
        default_locale/orgPath/filePath
        default/orgPath/filePath
```

where:

```
orgPath = subOrg1/subOrg2
        filePath = clientPath + serviceName
        clientPath = clientType/sub-clientType
```

The following are optional: Sub-org, Locale , Client Path , and Service Name . In the following example, `orgPath` and `filePath` are optional.

For example, given the following:

```
org = iplanet
locale = en
subOrg = solaris
clientPath = html/nokia/
serviceName = paycheck
```

the appropriate directory paths for the above are:

```
iplanet_en/solaris/html/nokia/paycheck
iplanet/solaris/html/nokia/paycheck
default_en/solaris/html/nokia/paycheck
default/solaris/html/nokia/paycheck
```

4 Copy all the JSP templates and authentication module configuration properties XML files from the default directory to the new directory.

`cp` *DistributedAuth-base*`/config/auth/default/*.jsp`
DistributedAuth-base`/config/auth/`*new_directory_path*

`cp` *DistributedAuth-base*`/config/auth/default/*.xml`
DistributedAuth-base`/config/auth/`*new_directory_path*

5 **(Optional) Modify the files in the new directory to suit your needs.**

- For information about customizing the `.jsp` files, see "Java Server Pages" on page 165.
- For information about customizing the `.xml` files, "XML Files" on page 167.

6 **Create a new `.WAR` file named `amauthdistui_deploy.war` from** *DistributedAuth-base*.

7 **Deploy `amauthdistui_deploy.war`.**

The web container administrator deploys the file in the remote web container.

Index

I

J

L

N

P

R

S

T

U

W