Sun Federated Access Manager 8.0 Developer's Guide

Beta

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

Part No: 820–3748–05
June 2008

List of Remarks

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 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 13
- "Sun Java Enterprise System Product Documentation" on page 15

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 Guideprovides 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 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 Guideprovides 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 2006Q4 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
- Support
- 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 Federated Access Manager 8.0 Developer's Guide*, and the part number is 820–3748.

◆ ◆ ◆ CHAPTER 1

Enhancing Remote Applications Using the Client Software Development Kit

The Sun Java™ System Federated Access Manager Client Software Development Kit (Client SDK) provides Java libraries for integrating access management functionality within stand-alone applications and web applications. The Client SDK can be used in remote applications to take advantage of Federated Access Manager services such as authentication, single sign-on (SSO), authorization, auditing and logging, and Security Assertion Markup Language (SAML). This chapter contains the following sections:

About the Client SDK

Remark 1–1 Reviewer **This chapter talks only about Java SDK. What about .NET and C? Is there stuff I should be writing about an SDK for those languages?**

The Federated Access Manager Client SDK contains Java packages and class files that can be used by developers to implement remote applications with Federated Access Manager services such as authentication, authorization, SSO, and SAML. The Client SDK is a streamlined version of the complete SDK installed with Federated Access Manager. The Client SDK includes only the client-side classes and configuration properties needed by remote applications to communicate with Federated Access Manager services. It is aimed at applications that use identity APIs at run time for authentication, SSO, policy evaluation and enforcement, and obtaining and setting user attributes. It is not for use by applications that perform policy

management or identity management (creation and deletion of entries). From a deployment point of view, the Client SDK offers the following:

- The Client SDK communicates directly with Federated Access Manager using XML (SOAP) over HTTP or HTTPS. In turn, Federated Access Manager communicates directly with the data store.
- The Client SDK does not require administrator credentials.
- Applications using the Client SDK can be deployed in demilitarized zones (DMZs), and a firewall can be placed between them and Federated Access Manager.
- The Client SDK includes samples to show how it can be used.

The packages that comprise the Client SDK include:

- `com.iplanet.am.sdk`
- `com.iplanet.am.util`
- `com.iplanet.sso`
- `com.sun.identity.authentication`
- `com.sun.identity.federation`
- `com.sun.identity.idm`
- `com.sun.identity.liberty.ws`
- `com.sun.identity.log`
- `com.sun.identity.policy`
- `com.sun.identity.policy.client`
- `com.sun.identity.saml`
- `com.sun.identity.saml2`
- `com.sun.identity.smt`
- `com.sun.identity.xacml`
- `com.sun.identity.wss`

Descriptions of these packages can be found in *Federated Access Manager 8.0 Java API Reference*. A complete listing of the classes that comprise the Client SDK can be found in the `ClientSDKClasses` file available on the OpenSSO web site.

Caution – It is recommended that developers don't call `com.iplanet.am.sdk`, `com.iplanet.am.util`, `com.sun.identity.policy`, and `com.sun.identity.sm` directly.

Using the Client SDK

[Remark 1–2 Reviewer: More real world type examples for this section would be great!!!] There are many ways to use the Client SDK. Following is a list of some of them.

- Build a proprietary application framework in which the Client SDK is a part. The Client SDK features can allow independence from policy agents.

- Access profile data to perform authentication and authorization beyond what is offered out-of-the-box.
- Allow authenticated and non-authenticated users access to a login process with a registration option that, if accepted, would create a user account.

Running the Client SDK Samples

Federated Access Manager comes with samples and source code that can help developers understand how the Client SDK classes can be implemented. The samples, acting as standalone applications, can be run on the command-line and in a web browser to see the function being performed. By looking at the provided sample source code you can understand how the Client SDK classes were used to perform the sample function.

`fam-client.zip` is the Client SDK sample ZIP and located in the `samples` directory of the inflated Federated Access Manager ZIP. After inflating `fam-client.zip` to its core `fam-client` directory, you will find two subdirectories:

- `sdk` contains the command line samples and source code. You must compile this before using the command line samples.
- `war` contains deployable WAR files comprised of the Client SDK and web-based samples.

The following sections further explain the two directories.

- "Web-based Samples" on page 19
- "Command Line Samples" on page 21

Web-based Samples

The web-based Client SDK samples are run by deploying a WAR file. The Client SDK WAR files are located in the `samples/fam-client/war` directory of the inflated Federated Access Manager download. They are:

- `fam-client-jdk15.war` requires Java Platform, Enterprise Edition 1.5.
- `fam-client-jdk14.war` requires Java 2, Standard Edition 1.4.2.

These WAR files contain the web-based samples and the Client SDK for use with them. Deploy either `fam-client-jdk14.war` or `fam-client-jdk15.war` to your web container, depending on the version of Java installed on the machine. After deploying, launching, and configuring the appropriate WAR, click the resulting link to proceed to the web-based samples Introduction page. This page contains links to the web-based samples.

Note – For more information on configuring the Client SDK, see "Using `AMConfig.properties` with Client SDK" on page 22.

The following table documents the web-based sample applications and their corresponding source file. Look in the `samples` directory for additional source code files not specifically called out below. The source files and directories noted in this table are linked to the version on the OpenSSO web site.

TABLE 1–1 Web-based Client SDK Samples

Sample	Function	Source
Service Configuration Sample Servlet	Retrieves and displays attributes of the entered service name	ServiceConfigServlet.java
User Profile (Attribute) Sample Servlet	Retrieves and displays the attributes that correspond to the entered user ID	UserProfileServlet.java
Policy Evaluator Client Sample Servlet	Retrieves from the Policy Service a policy decision that would be passed to a web agent for enforcement	PolicyClientServlet.java

TABLE 1–1 Web-based Client SDK Samples *(Continued)*

Sample	Function	Source
Single Sign-on Token Verification Servlet	Validates a session token and then displays the user profile associated with it	SSOTokenSampleServlet.java
Liberty ID-WSF 1.x Web Service Consumer Sample	Query and modify the Discovery Service and the Liberty Personal Profile Service	wsc Directory
Security Token Service (WS-Trust) Client Sample	Obtain security tokens from the Security Token Service	sts Directory

Command Line Samples

The command line samples are located in the `samples/fam-client/sdk` directory of the inflated Federated Access Manager download. These samples must be compiled before they can be used by running `scripts/compile-samples.sh`.

Caution – Be sure to run all the scripts discussed in this section from outside the `scripts` directory: `scripts/setup.sh`

The `README` in the `sdk` directory contains instructions on how to run the command line samples. The table documents the command line sample applications and their corresponding source file. Look in the `samples` directory for additional source code files not specifically called out below.

Note – The source files in this table are linked to the version on the OpenSSO web site.

TABLE 1–2 Command Line Client SDK Samples

Sample	Function	Source
`setup.sh`	Create `AMConfig.properties` and populate it with values based on your deployment.	Main.java

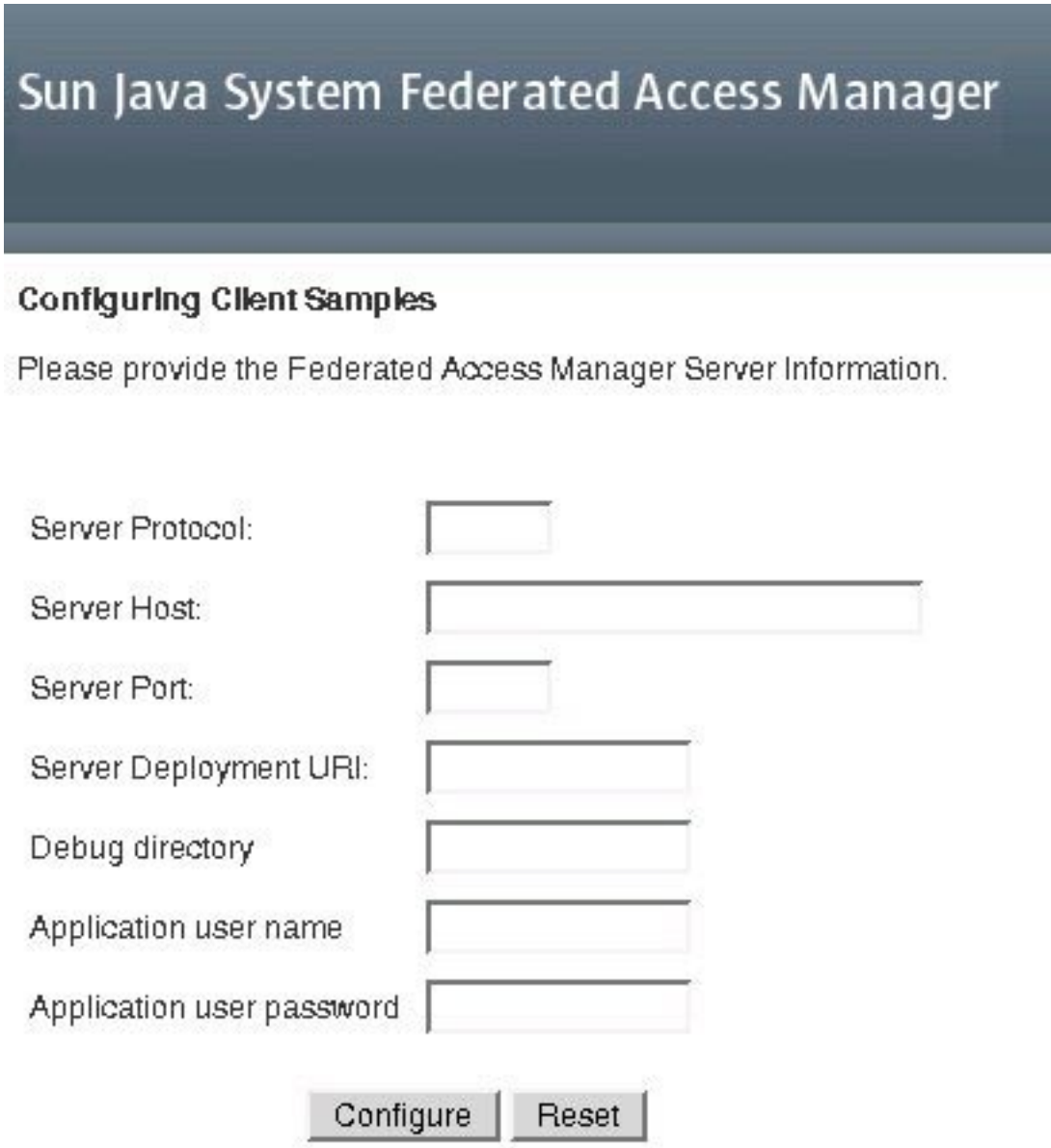
TABLE 1–2 Command Line Client SDK Samples *(Continued)*

Sample	Function	Source
Login.sh	Logs in and then logs out the user	Login.java
CommandLineSSO.sh	Demonstrates how to retrieve a user profile	CommandLineSSO.java
CommandLineIdrepo.sh	Perform operations on the Identity Repository; for example, create an identity, delete an identity and search or select an identity	idrepo Directory
CommandLineLogging.sh	Demonstrates the writing to a log a record of a successful authentication	logging Directory
SSOTokenSample.sh	Verifies a session token from a SSOTokenID input	SSOTokenSample.java
run-policy-evaluation-sample.sh	Returns a policy decision based on console created user and configured policy	policy Directory
run-xacml-client-sample.sh	Constructs a XACML request, makes an authorization query, receives the decision, and prints out the response	XACMLClientSample.java

Using AMConfig.properties with Client SDK

[Remark 1–3 Reviewer:] Although AMConfig.properties has been deprecated as the configuration data store for the Federated Access Manager application, the file is still used to store configuration data for the Client SDK. This AMConfig.properties points to the instance of Federated Access Manager that will be used by the Client SDK samples. After deploying and launching one of the sample WAR files (as discussed in "Web-based Samples" on page 19), a Client SDK configuration page is displayed.

Remark 1–4 Writer **Need new graphic header now says 'configure client SDK'**



Entering the appropriate values and clicking Configure creates an `AMConfig.properties` file under the home directory of the user running the web container. This value is indicated by the JDK system property `user.home`. When running the command line interface samples (as

discussed in "Command Line Samples" on page 21) AMConfig.properties is created in the samples/sdk/resources directory of the inflated Federated Access Manager ZIP.

Note – Both famclientsdk.jar and servlet.jar are required in the CLASSPATH of the machine on which the Client SDK is installed.

An AMConfig.properties file with the information needed to point to the remote Federated Access Manager server must be accessible to the Client SDK from the machine on which the client application is hosted. The AMConfig.properties created by the sample WAR can be modified for this purpose. The following sections explain how to do this.

Note – An AMConfig.properties file is also created and populated with values when the setup.sh script is run as discussed in "Command Line Samples" on page 21.

Federated Access Manager Properties for AMConfig.properties

Federated Access Manager properties used by the Client SDK are contained in the AMConfig.properties file generated by the Client SDK configured during installation. (See "Using AMConfig.properties with Client SDK" on page 22.) Additional properties can be added to this file as the client application can register for notification of changes to session and user attributes, and policy decisions. The following sections describe these properties.

Remark 1–5 Reviewer

Please review the properties in this section carefully. Let me know if any are missing or if any need to be removed. Also check the values: I modifed them from the opensso values I saw - mostly replacing opensso and amserver with fam. Finally, should the properties below match the properties in the AMConfig.properties file generated by the Client SDK configuration page?

Naming Properties

`com.iplanet.am.naming.url`

This is a required property. The value of this property is the URI of the Naming Service from which the Client SDK would retrieve the URLs of Federated Access Manager internal services. Example:

```
com.iplanet.am.naming.url=http://FAM_Host_Machine.domain_name:port
/fam/namingservice
```

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

This property can be used by any remote application developed with the Client SDK that wants failover in, for example, session validation or getting the service URLs. Example:

```
com.iplanet.am.naming.failover.url=http://FAM_Host_Machine.domain_name:port
/fam/failover
```

Debug Properties

`com.iplanet.services.debug.level`

Specifies the debug level. Values are:

- **`Off`** specifies that no debug information is recorded.
- **`error`** specifies that there should be no errors in the debug files. This level is recommended for production environments.
- **`warning`** is not a recommended value at this time.
- **`message`** alerts to possible issues using code tracing. Most Federated Access Manager modules use this level to send debug messages.

Caution – **`warning`** and **`message`** should not be used in production. They cause severe performance degradation and an abundance of debug messages.

`com.iplanet.services.debug.directory`

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

```
com.iplanet.services.debug.directory=/fam/debug
```

Notification URL Property

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

```
com.iplanet.am.notification.url=http://SDK_Host_Machine.domain_name:port
/fam/notificationservice
```

Security Credentials Properties

`com.sun.identity.agents.app.username`
User with permission to read Federated Access Manager configuration data. Default:

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

`com.iplanet.am.service.password`
Password of user with permission to read Federated Access Manager configuration data.

Note – Before running the Client SDK sample applications, you need to add `changeit` as a value for this property.

`com.iplanet.am.service.secret`
The encryption key used to encrypt the password. Example:

```
com.iplanet.am.service.secret=AQIC24u86rq9RRZGr/HN25OcIuO6w+ne+0lG
```

Encryption Properties

`am.encryption.pwd`
The encryption key used to decrypt service configuration passwords. Example:

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

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

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

`com.iplanet.security.encryptor`
Property to set the default encrypting class. Values are:

- `com.iplanet.services.util.JCEEncryption`
- `com.iplanet.services.util.JSSEncryption`

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 data if notification URL is not provided. Example:

```
com.iplanet.am.sdk.remote.pollingTime=1
```

Client Services Properties

These properties are defined by the Client SDK configuration page.

`com.iplanet.am.server.protocol`
Protocol of machine on which Federated Access Manager is deployed. Example:

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

`com.iplanet.am.server.host`
Name and domain of machine on which Federated Access Manager is deployed. Example:

```
com.iplanet.am.server.host=machine2.sun.com
```

`com.iplanet.am.server.port`
Port of machine on which Federated Access Manager is deployed. Example:

```
com.iplanet.am.server.port=8080
```

`com.iplanet.am.services.deploymentDescriptor`
URI of the deployed instance of Federated Access Manager. Example:

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

Cookie Property

`com.iplanet.am.cookie.name`
The name of the Federated Access Manager cookie. Example:

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

Session Service Properties

`com.iplanet.am.session.client.polling.enable`
A value of `true` or `false` enables or disables, respectively, client-side session polling.

`com.iplanet.am.session.client.polling.period`
Specifies the number of seconds in the polling period. Example

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

Certificate Database Properties

Remark 1–6 Writer **This only covers JSS. Most recently we've added some new properties for JCE/JSSE based provider to support SSL with client auth. We need to doc those properties too. This section should be tied to first two use cases in new use case section**

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

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

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

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

Policy Client Properties

`com.sun.identity.agents.server.log.file.name`
Specifies name of the client's policy log file. Example:

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

`com.sun.identity.agents.logging.level`
Specifies the granularity of logging to the client's policy log file.

- **NONE** is the default value. Nothing is logged.
- **ALLOW** logs allowed access decisions.
- **DENY** logs denied access decisions.
- **BOTH** logs allowed and denied access decisions.
- **DECISION**

`com.sun.identity.agents.notification.enabled`
A value of `true` or `false` enables or disables, respectively, notifications from Federated Access Manager for updating the client cache.

`com.sun.identity.client.notification.url`
Specifies the URL to which policy, session, and agent configuration notifications from Federated Access Manager are sent.

`com.sun.identity.agents.polling.interval`
Specifies the number of minutes after which an entry is dropped from the Client SDK cache. Example:

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

`com.sun.identity.policy.client.cacheMode`
Specifies the cache mode for the client policy evaluator. Values are:

- **`subtree`** specifies that the policy evaluator obtains policy decisions from the server for all the resources from the root of resource actually requested.
- **`self`** specifies that the policy evaluator obtains policy decisions from the server only for the resource actually requested.

`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` **[Remark 1–7 Reviewer: How do I reword this? No JES, right?]** Explicitly disables Java Enterprise System (JES) monitoring services in the sample client applications.

Remote Client SDK Property

`com.iplanet.am.sdk.package` If you want to use a remote instance of the Client SDK, set the value of this property to **`remote`**.

The default value is `ldap` if not explicitly defined.

Federation Properties

You must manually add these federation properties to `AMConfig.properties` as needed. They are not automatically placed in the file when generated.

com.sun.identity.wss.provider.plugins.AgentProvider

`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, 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.x`

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

`com.sun.identity.wss.provider.config.plugin=`
`com.sun.identity.wss.provider.plugins.AgentProvider`
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.AgentProvider`

`com.sun.identity.loginurl`
Used by the web services clients in Client SDK 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.

Initializing the `AMConfig.properties` Properties

[Remark 1–8 Reviewer: Please check these three sections and make sure they still work as documented.] When you configure the Client SDK (as documented in "Using `AMConfig.properties` with Client SDK" on page 22) you are minimally configuring it to communicate with a remote instance of Federated Access Manager. The properties listed in "Federated Access Manager Properties for `AMConfig.properties`" on page 24 can also be initialized. The following sections describe different ways in which these properties can be initialized.

Using the `AMConfig.properties` Properties File

You can set properties in the `AMConfig.properties` properties file created during installation. The properties are formatted as follows:

property_name=property_value

Note – The properties files must be in the CLASSPATH. If necessary, declare the Java Virtual Machine (JVM) option as follows:

`-Damconfig=`*properties_file_name*

Using the Java API

[Remark 1–9 Reviewer: Code sample still valid?] The Client SDK properties can be set programmatically using the class `com.iplanet.am.util.SystemProperties`. The following code sample illustrates how this can be accomplished.

EXAMPLE 1–1 Setting Client SDK Properties Programmatically

```
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');
        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 SDK Identity

[Remark 1–10 Reviewer: Changed this section. Please review carefully. How does the client send the username/PW that is stored in AMConfig?] Some Federated Access Manager components (such as SAML, user management, and policy) require an identity to be authenticated before the client application can read configuration data. The client can provide either a username and password that can be authenticated, or an implementation of the `com.sun.identity.security.AppSSOTokenProvider` interface. Either option will return a session token which the client can then use to access Federated Access Manager configuration data.

- "To Set Username and Password Properties" on page 34
- "To Set an SSO Token Provider" on page 34

To Set Username and Password Properties

The following properties in `AMConfig.properties` can be used to set the username and password. The authenticated username should have permission to read the Federated Access Manager configuration data.

- 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`.

Note – If a plain text password is a security concern, an encrypted password can be provided as the value of `com.iplanet.am.service.secret`. If an encrypted password is provided, the encryption key must also be provided as the value of `am.encryption.pwd`.

To Set an SSO Token Provider

[Remark 1–11 Reviewer: I don't see this property in AMConfig. Is it still there? Has this option changed? Shouldn't the implementation be used in the client app? Please explain.] Provide the implementation of the `com.sun.identity.security.AppSSOTokenProvider` interface as the value of the `com.sun.identity.security.AdminToken` property.

Client SDK Use Cases

Remark 1–12 Writer **Need to get these procedures**

This section contains the procedures for the following Client SDK use cases.

- Enabling the Client SDK to run against an SSL enabled instance of Federated Access Manager
- Enabling the Client SDK to run against an SSL enabled instance of Sun Directory Server
- how to enable Client SDK failover. Though this has been described on pager 15 via naming URL. It's good to list this as a separate section

SAE API

See SAE_README in home directory

Building Custom Web Applications

[Remark 1–13 Reviewer: It seems to me that this section should change. I need to speak with the appropriate engineer regard this.] [Remark 1–14 Writer: add details on what are needed in terms of jars , config files , properties files etc.] The Client SDK is contained in a small Java archive (JAR) named `famclientsdk.jar`. If using the Client SDK to write client applications, download (or retrieve from the `libraries/jars` directory of the Federated Access Manager ZIP) `famclientsdk.jar`, and include it in the class path for the application.

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

Remark 1–15 Writer **add details on what are needed in terms of jars , config files , properties files etc.**

Use this procedure for building identity-enabled web applications.

▼ To Build a Stand-Alone Application

1. **Install the Client SDK.**

   See "Running the Client SDK Samples" on page 19.

2. **Copy `servlet.jar` to the `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 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 the `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.

◆ ◆ ◆ CHAPTER 2

Using the Authentication Interfaces

This chapter provides information on the Sun Federated Access Manager Authentication Service application programming interfaces (API) and service provider interfaces (SPI). It contains the following sections:

Initiating Authentication with the Java Authentication API

The `com.sun.identity.authentication` package provides interfaces and classes that can be used by a Java application to access the Federated Access Manager Authentication Service. Through this access the application, running either locally or remotely toFederated Access Manager, can initiate an authentication process, submit required credentials and retrieve the single sign-on (SSO) session token (for an application or a user). When implemented, the authentication API starts the authentication process, and the Authentication Service responds with a set of requirements (user ID, password and the like). The appropriate credentials are returned to the Authentication Service. This back and forth communication between the custom application (with implemented API) and the Federated Access Manager Authentication Service continues until all requirements have been met. At that point, the client makes one final call to determine if authentication has been successful or has failed.

Note – There are authentication API for C applications. See *Sun Java System Federated Access Manager 8.0 C API Reference* for more information.

The first step in the code sequence for the authentication process is to instantiate the `com.sun.identity.authentication.AuthContext` class which will create a new `AuthContext`

object for each authentication request. Since Federated Access Manager can handle multiple organizations, `AuthContext` should be 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. The `getRquirements` method returns an array of `Callback` objects that correspond to the credentials the user must pass to the Authentication Service. These objects are requested by the authentication plug-ins, and are usually displayed to the user as 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 code 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 `submitRequirements()` method. The final step is 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.

The following code sample illustrates how to authenticate users with user name and password credentials and obtain the session token using `getSSOToken()`.

EXAMPLE 2–1 Authentication Code Sample

```
import com.iplanet.sso.SSOToken;
import com.sun.identity.authentication.AuthContext;

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.callback.UnsupportedCallbackException;
import javax.security.auth.login.LoginException;

public class TokenUtils {
    public static SSOToken getSessionToken(String orgName, String userId,
        String password) throws Exception {
        AuthContext ac = null;
        try {
            if (orgName == null || orgName.length() == 0) {
                orgName = "/";
            }
            ac = new AuthContext(orgName);
            ac.login();
        } catch (LoginException le) {
            le.printStackTrace();
            return null;
        }

        try {
            Callback[] callbacks = null;
            // Get the information requested by the plug-ins
```

EXAMPLE 2–1 Authentication Code Sample *(Continued)*

```
            if (ac.hasMoreRequirements()) {
                callbacks = ac.getRequirements();

                if (callbacks != null) {
                    addLoginCallbackMessage(callbacks, userId, password);
                    ac.submitRequirements(callbacks);

                    if (ac.getStatus() == AuthContext.Status.SUCCESS) {
                        System.out.println("Auth success");
                    } else if (ac.getStatus() == AuthContext.Status.FAILED) {
                        System.out.println("Authentication has FAILED");
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
        return ac.getSSOToken();
    }

    static void addLoginCallbackMessage(Callback[] callbacks, String userId,
        String password)
         throws UnsupportedCallbackException
    {
        int i = 0;
        try {
            for (i = 0; i < callbacks.length; i++) {
                if (callbacks[i] instanceof NameCallback) {
                    NameCallback nc = (NameCallback) callbacks[i];
                    nc.setName(userId);
                } else if (callbacks[i] instanceof PasswordCallback) {
                    PasswordCallback pc = (PasswordCallback) callbacks[i];
                    pc.setPassword(password.toCharArray());
                }
            }
        } catch (Exception e) {
             throw new UnsupportedCallbackException(callbacks[i],
                   "Callback exception: " + e);
        }
    }
}
```

Note – Because the Authentication Service is built on the Java Authentication and Authorization Service (JAAS) framework, the Authentication Service API can invoke any authentication modules written with the JAAS API as well as those built specifically for Federated Access Manager.

For a comprehensive listing of, and detailed information on, the Java API for authentication, see the *Federated Access Manager 8.0 Java API Reference*.

Writing Authentication Modules with the Java Authentication SPI

Federated Access Manager provides the `com.iplanet.authentication.spi` Java package to write Java-based authentication modules and plug them into the Authentication Service framework, allowing proprietary authentication providers to be managed using the administration console. The authentication module is created using the `com.iplanet.authentication.spi.AMLoginModule` class which implements the Java Authentication and Authorization Service (JAAS) `LoginModule` class.

Note – JAAS is a set of API that enables services to authenticate and enforce access controls upon users. It implements a Java version of the standard Pluggable Authentication Module (PAM) framework. Because of this architecture, any custom JAAS authentication module will work with the Authentication Service. For more information on the JAAS API, see the Java Authentication And Authorization Service Reference Guide. Additional information can be found at `http://java.sun.com/products/jaas/`

`com.iplanet.authentication.spi.AMLoginModule` provides methods to access the Authentication Service and the authentication module's configuration properties files. This class takes advantage of many built-in features of Federated Access Manager and scales well. Once created, the custom authentication module can be added to the list of authentication modules displayed by the Federated Access Manager console. The following steps represent an overview of the procedure to create an authentication module and plug it into the Federated Access Manager framework.

1. Create a module properties file.

   See "Creating an Authentication Module Configuration Properties File" on page 41.
2. Write a principal class.

   See "Writing the Principal Class" on page 43.
3. Implement the `LoginModule` interface.

See "Creating the Authentication Module" on page 43.

4. Add post processing tasks.

   See "Adding Post Processing Features" on page 44.

For a comprehensive listing of, and detailed information on, the Java SPI for authentication, see the *Federated Access Manager 8.0 Java API Reference*.

Creating an Authentication Module Configuration Properties File

The authentication module's configuration properties file is an XML file that defines the module's authentication requirements and login state information. The parameters in this file automatically and dynamically customize the authentication module's user interface, providing the means to initiate, construct and send the credential requests, in the form of login pages, to the Distributed Authentication User Interface. `Auth_Module_Properties.dtd` defines the data structure of the file.

Tip – Name the authentication module's configuration properties file using the same name as that of the authentication module's class (no package information) and use the extension `.xml`. Use this naming convention even if no states are required.

When an authentication process is invoked, values nested in the `Callbacks` element of the module's configuration properties file are used to generate login screens. The module controls the login process, and determines each concurring screen. The following configuration properties file for the LDAP authentication module illustrates this concept.

EXAMPLE 2–2 LDAP Authentication Module Configuration Properties File

```
<ModuleProperties moduleName="LDAP" version="1.0" >

    <Callbacks length="2" order="1" timeout="120"
     header="This server uses LDAP Authentication" >
        <NameCallback>
            <Prompt> User Name: </Prompt>
        </NameCallback>
        <PasswordCallback echoPassword="false" >
            <Prompt> Password: </Prompt>
        </PasswordCallback>
    </Callbacks>

    <Callbacks length="4" order="2" timeout="120"
     header="Change Password&lt;BR&gt;&lt;/BR&gt;#REPLACE#&lt;BR&gt;&lt;/BR&gt;" >
```

EXAMPLE 2–2 LDAP Authentication Module Configuration Properties File *(Continued)*

```
        <PasswordCallback echoPassword="false" >
            <Prompt>Old Password </Prompt>
        </PasswordCallback>
        <PasswordCallback echoPassword="false" >
            <Prompt> New Password </Prompt>
        </PasswordCallback>
        <PasswordCallback echoPassword="false" >
            <Prompt> Confirm Password </Prompt>
        </PasswordCallback>
        <ConfirmationCallback>
            <OptionValues>
                <OptionValue>
                    <Value> Submit </Value>
                </OptionValue>
                <OptionValue>
                    <Value> Cancel </Value>
                </OptionValue>
            </OptionValues>
        </ConfirmationCallback>
    </Callbacks>

    <Callbacks length="0" order="3" timeout="120"
     header=" Your password has expired. Please contact service desk to
     reset your password" error="true" />

    <Callbacks length="0" order="4" timeout="120" template="user_inactive.jsp"
     error="true"/>

</ModuleProperties>
```

The initial interface has two `Callback` elements corresponding to requests for the user identifier and password. When the user enters values, the following events occur:

- The values are sent to the module.
- The `process()` routine validates the values.

  If the module writer throws a `LoginException`, 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.
- If the user's password is expiring, the module writer sets the next page state to 2.

  Page state 2 requires the user to change a password. The `process()` routine is again called after the user submits the appropriate values.

Writing the Principal Class

After creating the authentication module's configuration properties file, write a class which implements `java.security.Principal` to represent the entity requesting authentication. For example, the constructor takes the username as an argument. If authentication is successful, the module will return this principal to the Authentication Service which populates the login state and session token with the information representing the user.

Creating the Authentication Module

Custom authentication modules extend the `com.sun.identity.authentication.spi.AMLoginModule` class and must implement the `init()`, `process()` and `getPrincipal()` methods. Other methods that can be implemented include `setLoginFailureURL()` and `setLoginSuccessURL()` which define URLs to which the user is sent based on a failed or successful authentication, respectively. To make use of the account locking feature with custom authentication modules, the `InvalidPasswordException` exception should be thrown when the password is invalid.

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

- The 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 `setAuthLevel()` method in the `init()` method implementation.

Information on implementing the three main methods is in the following sections:

- "Implementing the `init()` Method" on page 43
- "Implementing the `process()` Method" on page 44
- "Implementing the `getPrincipal()` Method" on page 44

Implementing the `init()` Method

`init()` is an abstract method that initializes the module with relevant information. This method is called by `AMLoginModule` prior to any other method calls. The method implementation should store the provided arguments for future use. It may peruse the `sharedState` to determine what information it was provided by other modules, and may also traverse through the `options` to determine the configuration parameters that will affect the module's behavior. The data can be ignored if the module being developed does not understand it.

Implementing the `process()` Method

`process()` is called to perform the actual authentication. For example, it may prompt for a user name and password, and then attempt to verify the credentials. If your module requires user interaction (for example, retrieving a user name and password), it should not do so directly. This method should invoke the `handle` method of the `javax.security.auth.callback.CallbackHandler` interface to retrieve and display the appropriate callbacks. The `AMLoginModule` then internally passes the callback values to the Distributed Authentication User Interface which performs the requested authentication.

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

- Perform the authentication and if successful, save the authenticated principal.
- Return `-1` if authentication succeeds.
- Throw an exception, such as `AuthLoginException`, if authentication fails or return the relevant state specified in the module's configuration properties file
- If multiple states are available to the user, the Callback array from a previous state may be retrieved by using the `getCallback()` method. The underlying login module keeps callback information from previous states until the login process is completed.
- If a module needs to substitute dynamic text (generate challenges, passwords or user identifiers) in the next state, use the `getCallback()` method to retrieve the callback for the next state, modify the text, and call `replaceCallback()` to update the array.
- Each authentication session will create a new instance of your module's Java class. The reference to the class will be released once the authentication session has either succeeded or failed.
- Any static data or reference to any static data in your module must be thread-safe.

Implementing the `getPrincipal()` Method

`getPrincipal()` should be called once at the end of a successful authentication session. This method retrieves the authenticated token string which will refer to the authenticated user in the Federated Access Manager environment. A login session is deemed successful when all pages in the module's configuration properties file have been sent and the module has not thrown an exception.

Adding Post Processing Features

The `com.sun.identity.authentication.spi.AMPostAuthProcessInterface` interface can be implemented for post processing tasks on authentication success, failure and logout using the methods `onLoginSuccess()`, `onLoginFailure()`, and `onLogout()`, respectively. The

Authentication Post Processing Classes are defined in the Core Authentication Service and configurable at several levels such as at the realm or role levels. Post processing tasks might include:

- Adding attributes to a user's session token 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.

Communicating Authentication Data as XML

Communication between applications and the Authentication Service is conducted with XML messages sent over `HTTP(s)`. The `remote-auth.dtd` is the template used to format the XML request messages sent to Federated Access Manager and to parse the XML return messages received by the external application. The `remote-auth.dtd` is in the *path-to-context-root*`/FAM/WEB-INF` directory.

- "XML Messages and `remote-auth.dtd`" on page 45
- "XML/HTTP(s) Interface for Other Applications" on page 47

XML Messages and `remote-auth.dtd`

The following sections contain examples of XML messages based on the `remote-auth.dtd`.

Note – The client application writes XML messages based on the `remote-auth.dtd` but, when the messages are sent, the Authentication API adds additional XML code to them. This additional XML is not illustrated in the following examples.

- "Authentication Request Message from Application" on page 45
- "Response Message from Federated Access Manager with Session Identifier and Callbacks" on page 46
- "Response Message from Application with User Credentials" on page 46
- "Authentication Status Message from Federated Access Manager With Session Token" on page 46

Authentication Request Message from Application

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

```
<?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>
```

Response Message from Federated Access Manager with Session Identifier and Callbacks

This example illustrates an affirmative response from Federated Access Manager that contains the session identifier for the original request (`authIdentifier`) as well as callback details.

```
<?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>
```

Response Message from Application with User Credentials

This example illustrates the client's response to Federated Access Manager. It contains the credentials input by the user to log in.

```
<?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>
```

Authentication Status Message from Federated Access Manager With Session Token

This example illustrates the message from Federated Access Manager specifying the user's successful authentication and the session token (`SSOToken`).

```
<?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>
```

XML/HTTP(s) Interface for Other Applications

Applications written in a programming language other than Java or C can also exchange authentication information with Federated Access Manager using the `XML/HTTP(s)` interface and the Authentication Service URL, `http://`*server_name.domain_name*`:`*port*`/`*service_deploy_uri*`/authservice`. An application can open a connection using the HTTP POST method. 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 Federated Access Manager.
- HTTP 1.1 compliant server implementation to receive XML-configured information from Federated Access Manager.
- An XML parser to interpret the data received from Federated Access Manager.

If contacting the Authentication Service directly through its URL, a detailed understanding of `remote-auth.dtd` will be needed for generating and interpreting the messages passed between the client and Federated Access Manager.

Working with the Authentication API Samples

Federated Access Manager comes with sample programs that demonstrate how to use the Authentication API to extend the functionality of the Authentication Service and authentication modules. Source code and a Makefile are provided for all sample programs. For some sample programs, additional supporting files are also included. The following sections contain information regarding these sample programs.

Java API Code Samples and Their Locations

The following tables describe the locations (on the various platforms) of all the files you need to implement the sample programs, and the variable names used for the default directories in the source code and Makefile.

- Table 2–1 summarizes file locations and variable names for Solaris Sparc/x86.
- Table 2–2 summarizes file locations and variable names for Linux.
- Table 2–3 summarizes file locations and variable names for Windows 2000.

TABLE 2–1 File Locations 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 Federated Access Manager is installed.	`install_root>/SUNWam`

TABLE 2–2 File Locations for Linux

Variable	Description	Location
Api_Sample_Dir	Directory that contains authentication API sample files	`<install_root>/sun/identity/samples/authentication/api`

TABLE 2–2 File Locations for Linux *(Continued)*

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

TABLE 2–3 File Locations 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 Federated Access Manager is installed.	`<install_root>`

The instructions for compiling and executing the sample programs are the same for all samples described in this section.

- "To Compile and Execute the Java API Samples" on page 49
- "To Configure SSL for Java API Samples" on page 50

▼ To Compile and Execute the Java API Samples

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

BASE_DIR: Enter the path to the directory where Federated 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 Federated 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 Federated Access Manager URL.

PASSWORD: Enter the Certificate DB Password.

3 **Copy** `AMConfig.properties` **from** *Config_Directory* **in the Federated Access Manager 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`**.**

6 **Copy all the property files from the** `locale` **directory in the Federated Access Manager host machine to the client machine.**

The `locale` directory on the server machine can be found under the *Product_Directory*.

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

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

9 **Compile the program.**

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

10 **Run the sample program.**

- On Solaris Sparc/x86 or Linux, run `gmake run`.
- On Windows 2000, run `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 Federated Access Manager 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 Java API Reference 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:

/FederatedAccessManager-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 49.

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:

/FederatedAccessManager-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 50.

2 **Compile and execute the sample code.**

See "To Compile and Execute the Java API Samples" on page 49

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:

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

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

/FederatedAccessManager-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:

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

Working with the Authentication SPI Samples

Federated Access Manager provides sample programs to demonstrate how to use the authentication SPI to extend authentication functionality. The following sections have more information.

- "Implementing a Custom Authentication Module" on page 53
- "Implementing the Authentication Post Processing SPI" on page 59
- "Generating an Authentication User ID" on page 63
- "Implementing A Pure JAAS Module" on page 66

Implementing a Custom Authentication Module

Federated 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 a Federated Access Manager deployment. All the files needed to compile, deploy and run the sample authentication module can be found in the following directory:

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

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

- "Compiling and Deploying the LoginModule program" on page 53
- "To Deploy the Login Module Sample Program" on page 54
- "Loading the Login Module Sample into Federated Access Manager" on page 56
- "Running the LoginModule Sample Program" on page 57

Note – The following are the default directories used in the sample exercises for the various platforms:

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

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

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

Compiling and Deploying the LoginModule program

If you are writing a custom authentication module based on the `AMLoginModule` SPI or JAAS, you can skip this section. Otherwise, after writing the sample Login Module, use these procedures to compile and deploy the sample found under *Federated Access Manager*`/samples/authentication/spi/providers`.

- "To Compile the Login Module" on page 53
- "To Deploy the Login Module Sample Program" on page 54
- "To Redeploy the `amserver.war` File" on page 54

▼ 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 in the following directory:
/FederatedAccessManager-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 Federated 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** */FederatedAccessManager-base*`/samples/authentication/spi/providers` **directory, run** `gmake`**.**

▼ To Deploy the Login Module Sample Program

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

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

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

▼ To Redeploy the `amserver.war` File

1 **In** */FederatedAccessManager-base*`/bin/amsamplesilent`**, set Deploy Level variable as follows:**
DEPLOY_LEVEL=21

2 **In** */FederatedAccessManager-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:

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

4 Restart the container instance.

- Web Server example:

```
/WebServer-base/
  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 Federated Access Manager

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

- "To Load the Login Module Using the Administration Console" on page 56
- "To Load the Login Module Using the Command Line" on page 56

▼ To Load the Login Module Using the Administration Console

1. **Login to the 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 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" on page 57
- "To Run the Login Module on Windows 2000" on page 58
- "To Deploy the Login Module" on page 58

▼ To Run the LoginModule on Solaris

1 **Use the following URL to log in to the 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: */FederatedAccessManager-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 **Run the `make` command from** */FederatedAccessManager-base*`\samples\authentication\spi\providers`**.**

▼ To Deploy the Login Module

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

2 **In the web container from which this sample will 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** */FederatedAccessManager-base*`\samples\authentication\spi\providers` **to** */FederatedAccessManager-base*`\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 the Authentication Post Processing SPI

The Authentication SPI includes the `AMPostAuthProcessInterface` which can be implemented for post processing tasks. 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. The Java API Reference for `AMPostProcessInterface` is available at:

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

Note – <PRODUCT_DIR> or */FederatedAccessManager-base* directory on different platforms:

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

- "To Compile the ISAuthPostProcess Sample Program on Solaris Sparc/x86 or Linux" on page 59
- "To Deploy the ISAuthPostProcess Sample Program on Solaris Sparc/x86 or Linux" on page 60
- "To Deploy the ISAuthPostProcess Sample Program on Windows 2000" on page 61
- "Configuring the Authentication Post Processing SPI" on page 61

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

Follow this procedure to compile the sample found under */FederatedAccessManager-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 Federated 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: */FederatedAccessManager-base*`/samples/authentication/spi/postprocess`.

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

▼ To Deploy the ISAuthPostProcess Sample Program on Solaris Sparc/x86 or Linux

1. **Copy** `ISAuthPostProcess.jar` **from** `JAR_DIR` **to** */FederatedAccessManager-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.

▼ To Deploy the ISAuthPostProcess Sample Program on Windows 2000

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

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`

More Information

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 61.

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 the 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 the 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 the 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.

Generating an Authentication User ID

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

- "To Compile the UserIDGeneratorSample on Solaris Sparc/x86 and Linux" on page 64
- "To Deploy the UserIDGeneratorSample Program on Solaris SPARC/x86 or Linux" on page 65
- "Configuring the UserIDGeneratorSample Program" on page 65
- "Compiling the UserIDGeneratorSample Program" on page 64

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`

Compiling the UserIDGeneratorSample Program

▼ To Compile the UserIDGeneratorSample on Solaris Sparc/x86 and 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 Compile the UserIDGeneratorSample on Windows 2000

1 Change to the `<install-root>\samples\authentication\spi\genuid` **directory.**

2 Run the `make` **command:**

Deploying the UserIDGeneratorSample Program

▼ To Deploy the UserIDGeneratorSample Program on Solaris SPARC/x86 or Linux

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`

▼ To Deploy the UserIDGeneratorSample Program on Windows 2000

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`

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 the UserIDGeneratorSample Program" on page 65
- "To Configure UserIDGeneratorSample for an Organization" on page 66
- "To Access an Authentication Module for an Organization" on page 66

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 61.

▼ 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.

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 2–4 File Locations for Solaris Sparc/x86

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

TABLE 2–5 File Locations for Linux

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

TABLE 2–6 File Locations for Windows 2000

Variable	Description	Location
Config_Directory	Directory that contains configuration files	`CONFIG_DIR = <install_root>\lib`
Product_Directory	Directory where Federated Access Manager is installed.	

▼ To Run the JAAS Module Sample on Solaris Sparc x86 or Linux

Before You Begin A sample configuration file `purejaassample.config` is provided for testing this sample. It contains only one entry: Sample. Sample is the value 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 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 Federated 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:

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

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 the Federated Access Manager host machine 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 **To compile, run the `gmake` command.**

5 **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

Before You Begin A sample configuration file `purejaassample.config` is provided for testing this sample. It contains only one entry: Sample. Sample is the value 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 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 Federated 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:

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

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

BASE: Enter the path to the directory where Federated 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 the Federated Access Manager host machine 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. **To compile, run the `make` command.**

5. **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 Java API Reference for Remote Client APIs, by default, in the following directory:

   /FederatedAccessManager-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

Enforcing Authorization with the Policy Service

Sun Java™ System Federated Access Manager enables organizations to control the usage of, and access to, their resources. This chapter provides information about how the Policy Service allows you to define, manage, and enforce policies towards that end. It contains the following sections:

About The Policy Service

The Policy Service provides the functionality to control access to web services and applications by providing authorization decisions based on defined and applicable *policies* or rules that define who or what is authorized to access a resource. In a single sign-on (SSO) environment, the Policy Service acts as authorization authority, providing authorization decisions that are enforced by a policy agent. The Policy Service acts as a Policy Administration Point (PAP) and a Policy Decision Point (PDP). As a PAP, it allows privileged users to create, modify, and delete access control policies. As a PDP, it provides access control decisions (after evaluating applicable policies) to a Policy Enforcement Point (PEP) which, in a Federated Access Manager environment, is a policy agent.

Note – For information on how the Policy Service works within a user session, see Chapter 5, "User Session and Single Sign-On Processes," in *Sun Federated Access Manager 8.0 Technical Overview*. Additional information is in *Chapter 4, "Authorization and the Policy Service," in Sun Java System Access Manager 7.1 Technical Overview*. More information on policy agents can be found in XXXX Policy Agents User's Guide XXXX.

About the Policy Service Interfaces

The Policy Service provides application programming interfaces (API) to administer policies and provide authorization decisions. It also provides service provider interfaces (SPI) to extend the Policy Service functionality. These interfaces are described in the following sections, organized by package names.

Note – Federated Access Manager also provides C API to enable external applications to connect to the Policy Service framework. For information about using the Policy C API, see *Chapter 3, "Policy Data Types and Functions," in Sun Java System Access Manager 7.1 C API Reference*. For a comprehensive listing of all Java interfaces and their usage, see the *Federated Access Manager 8.0 Java API Reference*.

The `com.sun.identity.policy` Package

`com.sun.identity.policy` contains classes for policy management and policy evaluation as described in the following sections.

Policy Management Classes

Policy management classes are used by privileged system administrators to programmatically add, look up, modify, replace and delete policies, and update the policy data store, if appropriate. Attempts by non-privileged users to manage policies will result in an exception and be logged. A valid session token is required to invoke any method provided by these classes. The key policy management classes are:

PolicyManager

`com.sun.identity.policy.PolicyManager` is the top-level administrator class for policy management in a specific realm. This class provides methods that enable the administrator to add, look up, modify, replace and delete policies. Only a privileged user with access to the policy data store and a valid session token can create a `PolicyManager` object. Some of the more widely used methods include:

`getPolicyNames()`	Retrieves all named policies created in the realm for which the `PolicyManager` object 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 realm for which the `PolicyManager` object was instantiated. If a policy with the same name already exists, it will be overwritten.
`removePolicy()`	Removes a policy from the realm for which the `PolicyManager` object was instantiated.
`replacePolicy()`	Overwrites a policy already defined in the realm for which the `PolicyManager` object was instantiated.

Policy

`com.sun.identity.policy.Policy` represents a policy definition with all its intended parts, including Rule(s), Subject(s), Condition(s), Referral(s) and Response Provider(s). The `Policy` object can be saved in the policy data store if the `addPolicy()` or `replacePolicy()` methods from the `PolicyManager` class are invoked. This class contains methods for adding, removing, replacing or retrieving any of the parts of a policy definition.

Policy Evaluation Classes

Policy Decision API is used to evaluate policy decision when a principal attempts an action on a resource. This section covers some key classes that provide Policy Evaluation API. Some classes are also provided to be used only by privileged users to test policy decisions applicable to other users.

Policy evaluation classes are used to evaluate the applicable policy when a principal attempts an action on a resource and send a determination on whether the principal will be allowed or denied access. The key policy evaluation classes are:

- PolicyEvaluator
- ProxyPolicyEvaluator
- PolicyEvent

Caution – 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. See "The `com.sun.identity.policy.client` Package" on page 75.

`PolicyEvaluator`

`com.sun.identity.policy.PolicyEvaluator` evaluates policy privileges and provides policy decisions. It provides methods to evaluate access to one resource or a hierarchy of resources, and supports both boolean and non-boolean type policies. A valid session token of the principal attempting access is required to invoke any method of this class. A `PolicyEvaluator` class is created by calling the constructor with a service name. Key public methods of this class include:

`isAllowed()`	Evaluates a policy associated with the given resource and returns a boolean-type value indicating an allow or deny decision.
`getPolicyDecision()`	Evaluates policies and returns a decision as to whether the associated principal can perform the specified actions on the specified resource.
`getResourceResults()`	A `ResourceResult` contains policy decisions regarding a particular protected resource and its sub resources. `getResourceResults()` obtains these policy decisions. Possible values for the scope of objects retrieved are `ResourceResult.SELF_SCOPE` (returns an object that contains the policy decision for the specified resource only), `ResourceResult.SUBTREE_SCOPE` (includes policy decisions for the specified resource and its sub-resources), and `ResourceResult.STRICT_SUBTREE_SCOPE` (returns an object that contains one policy decision regarding the `resourceName` only). For example, the `PolicyEvaluator` class can be used to display links for a list of resources to which an authenticated user has access. The `getResourceResults()` method can be used to retrieve a list of resources to which the user has access from a defined `resourceName` parameter — a URL in the form **`http://`***host.domain***`:`***port*. The resources are returned as a `PolicyDecision` object based on the user's 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 the `ActionDecision()` value contained in policy decisions.

ProxyPolicyEvaluator

`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 scope of administration. `com.sun.identity.policy.ProxyPolicyEvaluatorFactory` is the singleton class used to get `ProxyPolicyEvaluator` instances.

PolicyEvent

`com.sun.identity.policy.PolicyEvent` represents a policy event that could potentially change the current access status. A policy event is created and passed to 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.

The `com.sun.identity.policy.client` Package

The `com.sun.identity.policy.client` package contains classes that can be used by remote Java applications to evaluate policies and communicate with the Policy Service to get policy decisions. This package does not communicate with the policy data store therefore, use it when, for example, there is an intervening firewall. The package also maintains a local cache of policy decisions kept current either by a configurable time to live and/or notifications from the Policy Service.

The `com.sun.identity.policy.interfaces` Package

The `com.sun.identity.policy.interfaces` package contains SPI for writing custom plug-ins to extend the Policy Service. The classes are used by service developers and policy administrators who need to provide additional policy features as well as support for legacy policies.

`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.
`Referral`	Provides methods used to delegate the policy definition or evaluation of a selected resource (and its sub-resources) to another realm or policy server.

`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.
`ResponseProvider`	Defines an interface to allow pluggable response providers into the Federated Access Manager framework. Response providers are used to provide policy response attributes which typically provide attribute values from the user profile.
`Subject`	Provides methods to determine if an authenticated user is a member of the given subject.

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

The `com.sun.identity.policy.jaas` package provides classes for performing policy evaluation against Federated Access Manager using the Java Authentication and Authorization Service (JAAS) framework. JAAS is a set of APIs that enable services to authenticate and enforce access controls upon users. This package provides support for authorization only, making it possible to use JAAS interfaces to access the Policy Service. It contains the following implementations of JAAS classes:

- "`ISPermission`" on page 76
- "`ISPolicy`" on page 77

For more information see "Enabling Authorization Using the Java Authentication and Authorization Service" on page 77.

`ISPermission`

`com.sun.identity.policy.jaas.ISPermission` extends `java.security.Permission`, an abstract class for representing access to a resource. It represents the control of a sensitive operation, such as opening of a socket or accessing a file for a read or write operation. It does not grant permission for that operation, leaving that responsibility to the JAAS `AccessController` class which evaluates Federated Access Manager policy against the Policy Service.

Note – `ISPermission` covers the case when additional policy services are defined and imported provided they only have boolean action values as a JAAS permission only has a boolean result.

ISPolicy

`com.sun.identity.policy.jaas.ISPolicy` is an implementation of the JAAS abstract class `java.security.Policy` which represents the system policy for a Java application environment. It performs policy evaluation against the Policy Service instead of against the default file-based `PolicyFile`.

Enabling Authorization Using the Java Authentication and Authorization Service

Remark 3–1
Reviewer

Rewritten. Review carefully.

The Java Authentication and Authorization Service (JAAS) is a set of API that can determine the identity of a user or computer attempting to run Java code, and ensure that the entity has the right to execute the requested functions. After an identity has been determined using authentication, a `Subject` object, representing a grouping of information about the entity, is created. Whenever the `Subject` attempts a restricted operation or access, the Java runtime uses the JAAS `AccessController` class to determine which, if any, `Principal` (representing one piece of information established during authentication) would authorize the request. If the `Subject` in question contains the appropriate `Principal`, the request is allowed. If the appropriate `Principal` is not present, an exception is thrown.

Note – For more information see the JAAS Java API Reference.

In Federated Access Manager the custom implementation of the JAAS `java.security.Policy`, `com.sun.identity.policy.jaas.ISPolicy`, relies on the policy framework to provide policy evaluation for all Policy Service policies. Policy related to resources not under Federated Access Manager control (for example, system level resources) are evaluated using JAAS.

Note – Federated Access Manager policy does not control access to `com.sun.security.auth.PolicyFile`, the default JAAS policy store.

To enable authorization using JAAS in Federated Access Manager use the JAAS `java.security.Policy` API to reset policy during run time. In the sample code, the client application resets the policy to communicate with Federated Access Manager using `ISPolicy`. Federated Access Manager provides the support needed to define policy through `ISPermission`.

EXAMPLE 3–1 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;
               }
           });
       }
   }
```

Adding a Policy-Enabled Service to Federated Access Manager

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

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>
```

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

```
                    <BooleanFalseValue i18nKey="deny">deny</BooleanFalseValue>
                </BooleanValues>
        </AttributeSchema>

        <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>
```

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.

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

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

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

2 **Copy the properties file to the `locale` directory of the Federated Access Manager host machine.**

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

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

`/etc/opt/SUNWam/config/xml/amWebAgent.xml` (Solaris)
`/etc/opt/sun/identity/config/xml/amWebAgent.xml`(Linux and HP-UX)
Federated Access Manager`\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>
```

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

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

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

Using the Policy Code Samples

Federated Access Manager provides policy code samples to perform the following tasks:

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

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

Solaris Platform */FederatedAccessManager-base*`\samples\policy`

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

This section contains the following information regarding the samples.

Use Cases Illustrated by Policy Code Samples

Each of the following sections describes a sequence of steps you must take 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.

Policy Evaluation

This section contains the following procedures:

▼ 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 92.

▼ 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 85.
2. **Develop custom subjects, conditions, and referrals.**
   See "Developing Custom Subjects, Conditions, Referrals, and Response Providers" on page 86.
3. **Develop and run a Policy evaluation program.**
   See "Developing and Running a Policy Evaluation Program" on page 92.

▼ 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 85.
2. **Add a Policy-enabled service to Access Manager.**
   See "Adding a Policy-Enabled Service to Federated Access Manager" on page 79.
3. **Create policies for the new service.**
   See "Creating Policies for a New Service" on page 91.
4. **Develop and run a Policy evaluation program.**
   "Developing and Running a Policy Evaluation Program" on page 92.

▼ 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 85.
2. **Add a Policy-enabled service to Access Manager.**
   See "Adding a Policy-Enabled Service to Federated Access Manager" on page 79.
3. **Develop custom subjects, conditions, and referrals.**
   See "Developing Custom Subjects, Conditions, Referrals, and Response Providers" on page 86.
4. **Create policies for the new service.**
   See "Creating Policies for a New Service" on page 91.

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

See "Developing and Running a Policy Evaluation Program" on page 92.

Using `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" on page 85
- "To Programmatically Construct Policies" on page 85

▼ To Use `amadmin` to Create Policies for the Sample Service

This sequence creates policies for `SampleWebService`.

1 **Compile the Policy code samples.**

See "Compiling the Policy Code Samples" on page 85.

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

See "Developing and Running a Policy Evaluation Program" on page 92.

▼ To Programmatically Construct Policies

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

1 **Compile the Policy code samples.**

See "Compiling the Policy Code Samples" on page 85.

2 **Programmatically construct policies.**

See "Programmatically Constructing Policies" on page 94.

Compiling the Policy Code Samples

Samples can be run on Solaris, Linux, HP-UX, and Windows platforms. In the sample files, root suffix `DN`s are specified as `dc=example,dc=com`. Substitute the root suffix with the actual root suffix of your Federated 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 to the directory where Federated 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.**

Developing Custom Subjects, Conditions, Referrals, and Response Providers

Federated 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 Federated 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 Federated 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:

/FederatedAccessManager-base/SUNWam/samples/policy

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

EXAMPLE 3–3 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"
                      i18nKey="iplanet-sample-referral"
                      interfaceName="Referral"
                      name="SampleReferral" >
        </PluginSchema>
        <PluginSchema className="SampleResponseProvider"
                      i18nFileName="amPolicy"
                      i18nKey="iplanet-sample-responseprovider"
                      interfaceName="ResponseProvider"
                      name="SampleResponseProvider" >
        </PluginSchema>
    </Service>
```

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

```
</ServicesConfiguration>
```

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

EXAMPLE 3–4 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>

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

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

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

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

```
    <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 /FederatedAccessManager-base/samples/policy
      /FederatedAccessManager-base/bin/amadmin
--runasdn "uid=amAdmin,ou=People,default_org,root_suffix
--password password
--schema amPolicy_mod.xml
      /FederatedAccessManager-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 /FederatedAccessManager-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 */FederatedAccessManager-base*`/lib`.

6 **Restart Federated Access Manager.**

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

Federated 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 */FederatedAccessManager-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 85 and "Developing Custom Subjects, Conditions, Referrals, and Response Providers" on page 86 for instructions.

1 Run the following command:

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

2 Run the following command:

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

You can verify the newly added policies in Administration Console.

Developing and Running a Policy Evaluation Program

Federated 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`. Federated 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.

This section contains the following procedures.

- "To Set Policy Evaluation Properties" on page 93
- "To Run a Policy Evaluation Program" on page 94

▼ 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

Federated 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 3–5 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 3–5 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 3–5 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 3–5 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 3–5 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 85 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 Federated Access Manager.

◆ ◆ ◆ CHAPTER 4

Tracking Session Data for Single Sign-On

The Federated Access Manager Session Service maintains information about an authenticated user's session across all web applications in a single sign-on (SSO) environment. This chapter describes the interfaces used to track session data for purposes of SSO and related sample code. It includes the following sections:

A Simple Single Sign-On Scenario

In an SSO scenario, a user logs in to a protected resource. Once the user has successfully authenticated to Federated Access Manager, a user session is created and stored in Federated Access Manager memory. The user object uses browser cookies or URL query parameters to carry a session identifier. Each time the user requests access to another protected resource, the new application must verify the user's identity. It does not ask the user to present credentials. Instead, the application uses the session identifier and the Session Service interfaces to retrieve the user's session information from Federated Access Manager. If it is determined from the session information that the user has already been authenticated and the session is still valid, the new application allows the user access to its data and operations. If the user is not authenticated, or if the session is no longer valid, the requested application prompts the user to present credentials a second time. Until logging out, this scenario is played out every time the user accesses a protected resource in the SSO environment. For more detailed information about user sessions and SSO, see Chapter 5, "User Session and Single Sign-On Processes," in *Sun Federated Access Manager 8.0 Technical Overview*.

Inside a User Session

A user session is, more specifically, a data structure created by the Session Service to store information about a user session. Cookies are used to store a token that uniquely identifies the session data structure. A session data structure contains attributes and properties that define the user's identity and time-dependent behaviors (for example, the maximum time before the session expires).

Note – The values of most of these attributes and properties are set by services other than the Session Service (primarily, the Authentication Service). The Session Service only provides storage for session information and enforces some of the time-dependent behavior; for example, invalidating and destroying sessions which exceed their maximum idle time or maximum session time.

The following sections contain information about the session attributes and properties contained in the session data structure.

- "Session Attributes" on page 102
- "Protected And Custom Properties" on page 103

Session Attributes

The session data structure contains the following fixed attributes:

`UUID`	This universal, unique session identifier is an opaque, global string that programmatically identifies a specific session data structure. With this identifier, a resource is able to retrieve session information.
`ClientDomain`	This is the DNS domain in which the client is located.
`ClientID`	This is the user DN or the application's principal name.
`Type`	This is specifies the type of client: USER or APPLICATION.
`State`	This is the state of the session: VALID, INVALID, DESTROYED or INACTIVE.
`maxIdleTime`	This is the maximum time in minutes without activity before the session will expire and the user must reauthenticate.
`maxSessionTime`	This is the maximum time in minutes before the session expires and the user must reauthenticate.
`maxCachingTime.`	This is the maximum time in minutes before the client contacts Identity Server to refresh cached session information
`latestAccessTime`	This refers to the last time the user accessed the resource.

`creationTime`	This is the time at which the session token was set to a valid state.

Protected And Custom Properties

The session data structure also contains an extensible set of protected (or core) properties and custom properties. Protected properties are set by Federated Access Manager and can only be modified by Federated Access Manager (primarily the Authentication Service). Custom properties are set and modified remotely by any application that knows the session identifier.

Note – The session property implementation can be extended to provide the private property scope for each of the clients participating in the SSO environment. This addresses the requirement of having an independent property space for each client to store and retrieve its own session properties without interference from other clients sharing the same user session.

See the following sections for more information.

- "Protected Properties" on page 103
- "Custom Properties" on page 104

Protected Properties

The current protected properties used are:

`Organization`	This is the DN of the organization to which the user belongs.
`Principal`	This is the DN of the user.
`Principals`	This is a list of names to which the user has authenticated. (This property may have more then one value defined as a pipe separated list.)
`UserId`	This is the user's DN as returned by the module, or in the case of modules other than LDAP or Membership, the user name. (All Principals must map to the same user. The `UserId` is the user DN to which they map.)
`UserToken`	This is a user name. (All Principals must map to the same user. The UserToken is the user name to which they map.)
`Host`	This is the host name or IP address for the client.
`authLevel`	This is the highest level to which the user has authenticated.
`AuthType`	This is a pipe separated list of authentication modules to which the user has authenticated (for example, `module1|module2|module3`).
`Role`	Applicable for role-based authentication only, this is the role to which the user belongs.

`Service`	Applicable for service-based authentication only, this is the service to which the user belongs.
`loginURL`	This is the client's login URL.
`Hostname`	This is the host name of the client.
`cookieSupport`	This attribute contains a value of true if the client browser supports cookies.
`authInstant`	This is a string that specifies the time at which the authentication took place.
`SessionTimedOut`	This attribute contains a value of true if the session has timed out.

Custom Properties

The custom properties currently used are:

`clientType`	This is the device type of the client browser.
`Locale`	This is the locale of the client.
`CharSet`	This is the determined character set for the client.

About the Session Service Interfaces

All Federated Access Manager services (except for the Authentication Service) require a valid session identifier (programmatically referred to as `SSOToken`) to process an HTTP request. External applications developed using the Session Service interfaces and protected by a policy agent also require an `SSOToken` to determine access. The `SSOToken` is an encrypted, unique string that identifies a specific session data structure stored by Federated Access Manager. If the `SSOToken` is known to a Federated Access Manager service or an external protected resource such as an application, the service or application can access all user information and session data stored in the session data structure it identifies. After successful authentication, the `SSOToken` is transported using cookies or URL parameters, allowing participation in SSO.

The Session Service provides Java interfaces to allow Federated Access Manager services and external applications to participate in the SSO functionality. The `com.iplanet.sso` package contains the tools for creating, destroying, retrieving, validating and managing session data structures and session identifiers. All external applications wishing to participate in the SSO solution must be developed using this API. In the case of a remote application, the invocation is forwarded to Federated Access Manager by the client libraries using XML messages over HTTP(S).

Note – Federated Access Manager also includes an API for session management in C applications. For information see Chapter 4, "Single Sign-On Data Types and Functions," in *Sun Java System Federated Access Manager 8.0 C API Reference*.

The following sections contain more specific information about the contents of `com.iplanet.sso`.

- "`SSOTokenManager`" on page 105
- "`SSOToken`" on page 106
- "`SSOTokenListener`" on page 109

For a comprehensive listing of all Java interfaces and their usage, see the *Federated Access Manager 8.0 Java API Reference*.

SSOTokenManager

The `SSOTokenManager` class contains the methods needed to get, validate, destroy and refresh the session identifiers that are programmatically referred to as the `SSOToken`. To obtain an instance of `SSOTokenManager`, call the `getInstance()` method. Once instantiated, `SSOTokenManager` can be used to create an `SSOToken` object using one of the forms of the `createSSOToken()` method. The `destroyToken()` method is called to invalidate and delete a token when its corresponding session has ended. Either the `isValidToken()` and `validateToken()` methods can be called to verify whether a token is valid (asserting successful authentication). `isValidToken()` returns true or false depending on whether the token is valid or invalid, respectively. `validateToken()` throws an exception only when the token is invalid; nothing happens if the token is valid. The `refreshSession()` method resets the idle time of the session. Example 4–1 illustrates one way in which the `SSOTokenManager` class can be used.

EXAMPLE 4–1 `SSOTokenManager` Code Sample

```
try {

        /* get an instance of the SSOTokenManager */

SSOTokenManager ssoManager = SSOTokenManager.getInstance();

        /* The request here is the HttpServletRequest. Get
        /* SSOToken for session associated with this request. */

SSOToken ssoToken = ssoManager.createSSOToken(request);

        /* use isValid method to check if token is valid or not.
        /* This method returns true for valid token, false otherwise. */
```

EXAMPLE 4–1 `SSOTokenManager` Code Sample *(Continued)*

```
if (ssoManager.isValidToken(ssoToken)) {

        /* If token is valid, this information may be enough for
        /* some applications to grant access to the requested
        /* resource. A valid user represents a user who is
        /* already authenticated. An application can further
        /* utilize user identity information to apply
        /* personalization logic .*/

} else {

        /* Token is not valid, redirect the user login page. */

}

        /* Alternative: use of validateToken method to check
        /* if token is valid */

try {
ssoManager.validateToken(ssoToken);

        /* handle token is valid */

} catch (SSOException e) {

        /* handle token is invalid */

}

        /*refresh session. idle time should be 0 after refresh. */

ssoManager.refreshSession(ssoToken);

} catch (SSOException e) {

        /* An error has occurred. Do error handling here. */

}
```

SSOToken

The `SSOToken` interface represents the session identifier returned from the `createSSOToken()` method, and is used to retrieve session data such as the authenticated principal name,

authentication method, and other session information (for example, session idle time and maximum session time). The `SSOToken` interface has methods to get predefined session information such as:

- `getProperty()` is used to get any information about the session, predefined or otherwise (for example, information set by the application).
- `setProperty()` can be used by the application to set application-specific information in the session.
- `addSSOTokenListener()` can be used to set a listener to be invoked when the session state has become invalid.

Caution – The methods `getTimeLeft()` and `getIdleTime()` return values in seconds while the methods `getMaxSessionTime()` and `getMaxIdleTime()` return values in minutes.

The following code samples illustrate ways to use the interface.

- Example 4–2 illustrates one way in which the `SSOToken` interface can be used.
- Example 4–3 illustrates how to use the `getTokenID()` method to create a cookie from a session token in order to allow SSO to work on protected resources not residing on the same server as Federated Access Manager.

EXAMPLE 4–2 `SSOToken` Code Sample

```
        /* get http request output stream for output */

ServletOutputStream out = response.getOutputStream();

        /* get the sso token from http request */

SSOTokenManager ssoManager = SSOTokenManager.getInstance();
SSOToken ssoToken = ssoManager.createSSOToken(request);

        /* get the sso token ID from the sso token */

SSOTokenID ssoTokenID = ssoToken.getTokenID();
out.println("The SSO Token ID is "+ssoTokenID.toString());

        /* use validate method to check if the token is valid */

try {
ssoManager.validateToken(ssoToken);
out.println("The SSO Token validated.");

} catch (SSOException e) {
```

EXAMPLE 4–2 SSOToken Code Sample *(Continued)*

```
out.println("The SSO Token failed to validate.");
}

        /* use isValid method to check if the token is valid */

if (!ssoManager.isValidToken(token)) {
out.println("The SSO Token is not valid.");
} else {

        /* get some values from the SSO Token */

java.security.Principal principal = ssoToken.getPrincipal();
out.println("Principal name is "+principal.getName());

String authType = ssoToken.getAuthType();
out.println("Authentication type is "+authType);

int authLevel = ssoToken.getAuthLevel();
out.println("Authentication level is "+authLevel);

long idleTime = ssoToken.getIdleTime();
out.println("Idle time is "+idleTime);

long maxIdleTime = ssoToken.getMaxIdleTime();
out.println("Max idle time is "+maxIdleTime);

long maxTime = token.getMaxSessionTime();
out.println("Max session time is "+maxTime);

String host = ssoToken.getHostName();
out.println("Host name is "+host);

        /* host name is a predefined information of the session,
        /* and can also be obtained the following way */

String hostProperty = ssoToken.getProperty("HOST");
out.println("Host property is "+hostProperty);

        /* set application specific information in session */

String appPropertyName = "appProperty";
String appPropertyValue = "appValue";
ssoToken.setProperty(appPropertyName, appPropertyValue);

        /* now get the app specific information back */
```

EXAMPLE 4–2 SSOToken Code Sample *(Continued)*

```
String appValue = ssoToken.getProperty(appPropertyName);
if (appValue.equals(appPropertyValue)) {
out.println("Property "+appPropertyName+",
 value "+appPropertyValue+" verified to be set.");
} else {
out.println("ALERT: Setting property "+appPropertyName+" failed!");

}

}
```

EXAMPLE 4–3 getTokenID() Code Sample

```
        // Get SSOToken string

String strToken = null;
strToken = getSSOToken().getTokenID().toString();

        // Set it to response as cookies

String s = strToken;
String ssotokencookiename = "iPlanetDirectoryPro";
String ssotokencookiedomain = ".mydomain.com.tw";
String ssotokencookiepath = "/";
String gt = "/welcomepage.jsp";

Cookie cookie = new Cookie(ssotokencookiename,s);
cookie.setDomain(ssotokencookiedomain);
cookie.setPath(ssotokencookiepath);
response.addCookie(cookie);
response.sendRedirect(gt);
```

SSOTokenListener

Remark 4–1 Writer **Sent email regarding docing this and SSOTokenID**

The SSOTokenListener class allows the application to be notified when a SSOToken has become invalid — for example, when a session has timed out.

Using the SSO Code Samples

Federated 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 a Federated Access Manager session cookie to be 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 Federated Access Manager

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 Federated Access Manager is installed.
`CLASSPATH`	Specify the directory where the JAR files are installed. Example: *FederationManager-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 */FederatedAccessManager-base*`/SUNWam/samples/sso`**, run the** `gmake` **command.**

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

4. **Update the web container classpath.**

   a. **Create a web server administrator 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 Federated Access Manager.**

7. **Log in to the Federated 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 domain 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 Federated Access Manager Client API 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 Client API are installed in a directory named `opt`. For information on installing the Client APIs, see Chapter 1, "Enhancing Remote Applications Using the Client Software Development Kit."

1 **In the directory** */FederatedAccessManager-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 Web Server.**

If Federated Access Manager 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 Federated Access Manager Client APIs are installed.**

4 **Log in to the Federated Access Manager console.**

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 Federated Access Manager console.**

Because no log out link exists in the sample servlet, you must use a browser to access the 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 Federated Access Manager Client API before you can run a sample program on the remote client command line. For more information on using the Client APIs, see Chapter 1, "Enhancing Remote Applications Using the Client Software Development Kit."

When you run an 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 Federated 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** *Federated Access Manager*`/SUNWam/samples/sso`**, run the** `gmake` **command.**

2 **Modify the script** *Federated Access Manager*`/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** *Federated Access Manager*`/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 AQIC5wM2LY4SfcwbdpgWuB38NA26klnTJlLPknN8t0fPVY=
    Property: Company is - Sun Microsystems
    Property: Country is - USA
    SSO Token Validation test Succeeded
```

2 **In the** *Federated Access Manager*`/SUNWam/samples/sso` **directory, execute the** `run` **command:**

```
run AQIC5wM2LY4SfcwbdpgWuB38NA26klnTJlLPknN8t0fPVY=
```

The following result is displayed:

```
SSO "AQIC5wM2LY4SfcwbdpgWuB38NA26klnTJlLPknN8t0fPVY="
    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

Federated 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 Federated 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 Federated Access Manager access valid only within that process or VM.

◆ ◆ ◆ CHAPTER 5

Implementing the Liberty Alliance Project Identity-Federation Framework

Sun Java™ System Federated Access Manager has a robust framework for implementing federated identity infrastructures. It provides interfaces, based on the Liberty Alliance Project Identity-Federation Framework (Liberty ID-FF) for creating, modifying, and deleting circles of trust, service providers, and identity providers as well as samples to get you started. This chapter covers the following topics:

About the Liberty ID-FF

The Liberty Alliance Project addresses the need for an open industry standard for federated identity management. The concept of a federated identity begins with the notion of a virtual identity. On the internet, one person might have a multitude of accounts set up to access various business, community and personal service providers — each of these providers is said to have a virut. For example, the person might have used different names, user identifiers, passwords or preferences to set up accounts for a news portal, a bank, a retailer, and an email provider, respectively; thus the person has a different virtual identity for each web site has a different virtua. A *local identity* refers to the set of attributes that an individual might have with each of these service providers. These attributes uniquely identify the individual with that particular provider and can include a name, phone number, passwords, social security number, address, credit records, bank balances or bill payment information. Because the internet is fast becoming the prime vehicle for business, community and personal interactions, it has become necessary to fashion a system for online users to link their local identities, enabling them to have one *network identity*.

Understanding Federation

The umbrella term **federation** encompasses both *identity federation* and *provider federation*. The concept of *identity federation* begins with the notion of virtual identity. On the internet, one person might have a multitude of accounts set up to access various business, community and personal service providers; for example, the person might have used different names, user identifiers, passwords or preferences to set up accounts for a news portal, a bank, a retailer, and an email provider. A *local identity* refers to the set of attributes that an individual might have with each of these service providers. These attributes uniquely identify the individual with that particular provider and can include a name, phone number, passwords, social security number, address, credit records, bank balances or bill payment information. Because the internet is fast becoming the prime vehicle for business, community and personal interactions, it has become necessary to fashion a system for online users to link their local identities, enabling them to have one *network identity*. This system is *identity federation*. Identity federation allows a user to associate, connect or bind the local identities they have configured with multiple service providers. A *federated identity* allows users to login at one service provider's site and move to an affiliated service provider site without having to re-authenticate or re-establish their identity.

The concept of *provider federation* as defined in a federation-based environment begins with the notion of a security domain (referred to as a *circle of trust* in Federated Access Manager). A *circle of trust* is a group of service providers (with at least one identity provider) that agree to join together to exchange user authentication information using open—standards and technologies. Once a group of providers has been federated within a circle of trust, authentication accomplished by the identity provider in that circle is honored by all affiliated service providers. Thus, single sign-on (SSO) can be enabled amongst all membered providers as well as identity federation among users. For more information on the federation process in Federated Access Manager, see the *Sun Federated Access Manager 8.0 Technical Overview*. The following sections contain information on the federation specifications implemented by Federated Access Manager.

Federated Access Manager supports the *Liberty Alliance Identity Federation Framework 1.2 Specifications* and the *WS-Federation 1.1 Metadata*.

Customizing the Federation Graphical User Interface

The Federation Service uses JavaServer Pages™ (JSP™) to define its look and feel. *JSP* are HTML files that contain additional code to generate dynamic content. More specifically, a JavaServer page contains HTML code to display static text and graphics, as well as application code to generate information. When the page is displayed in a web browser, it contains both the static HTML content and, in the case of the Federation component, dynamic content retrieved through calls to the Federation API. An administrator can customize the look and feel of the interface by changing the HTML tags in the JSP but the invoked APIs must not be changed.

The JSP are located in
/path-to-context-root`/fam/web-src/services/config/federation/default`. The files in this

directory provide a default interface to the Federation Service. To customize the pages for a specific organization, this default directory can be copied and renamed to reflect the name of the organization (or any value). This directory would then be placed at the same level as the default directory, and the files within this directory would be modified as needed. The following table lists the JSP including details on what each page is used for and the invoked APIs that cannot be modified.

JSP Name and Implemented APIs	Purpose
■ `CommonLogin.jsp` Invoked APIs are: ■ `LibertyManager.getLoginURL(request)` ■ `LibertyManager.getInterSiteURL(request)` ■ `LibertyManager.getIDPList(providerID)` ■ `LibertyManager.getNewRequest(request)` ■ `LibertyManager.getSuccintID(idpID)` ■ `LibertyManager.cleanQueryString(request)`	Displays a link to the local login page as well as links to the login pages of the trusted identity providers. This page is displayed when a user is not logged in locally or with an identity provider. The list of identity providers is obtained by using the `getIDPList(hostedProviderID)` method.
■ `Error.jsp`	Displays an error page when an error has occurred. No APIs are invoked.
■ `Federate.jsp` Invoked APIs are: ■ `LibertyManager.isLECPProfile(request)` ■ `LibertyManager.getAuthnRequestEnvelope(request)` ■ `LibertyManager.getUser(request)` ■ `LibertyManager.getProvidersToFederate(providerID,userDN)`	Displays when a user clicks a federate link on a provider page. Contains a drop-down of all providers with which the user is not yet federated. This list is constructed by using the `getProvidersToFederate(userName,providerID)` method.
■ `FederationDone.jsp` Invoked API is: ■ `LibertyManager.isFederationCancelled(request)`	Displays the status of a federation (success or cancelled). This page checks the status by using the `isFederationCancelled(request)` method.

JSP Name and Implemented APIs	Purpose
■ `Footer.jsp`	Displays a branded footer that is included on all the pages. No APIs are invoked.
■ `Header.jsp`	Displays a branded header that is included on all the pages. No APIs are invoked.
■ `ListOfCOTs.jsp` Invoked API is: ■ `LibertyManager.getListOfCOTs(providerID)`	Displays a list of circles of trust. When a user is authenticated by an identity provider and the service provider belongs to more than one circle of trust, the user is shown this JSP and is prompted to select an authentication domain as their preferred domain. In the case that the provider belongs to only one domain, this page will not be displayed. The list is obtained by using the `getListOfCOTs(providerID)` method.
■ `LogoutDone.jsp` Invoked API is: ■ `LibertyManager.isLogoutSuccess(request)`	Displays the status of the local logout operation.
■ `NameRegistration.jsp` Invoked APIs are: ■ `LibertyManager.getUser(request)` ■ `LibertyManager.getRegisteredProviders(userDN)`	Displays when the Name Registration link is clicked on a provider page. When a federated user chooses to register a new Name Identifier from a service provider to an identity provider, this JSP is displayed.
■ `NameRegistrationDone.jsp` Invoked APIs are: ■ `LibertyManager.isNameRegistrationSuccess(request)` ■ `LibertyManager.isNameRegistrationCanceled(request)`	Displays the status of `NameRegistration.jsp`. When finished, this page is displayed.
■ `Termination.jsp` Invoked APIs are: ■ `LibertyManager.getUser(request)` ■ `LibertyManager.getFederatedProviders(userDN)`	Displays when a user clicks a defederate link on a provider page. Contains a drop-down of all providers to which the user has federated and from which the user can choose to defederate. The list is constructed by using the `getFederatedProviders(userName)` method, which returns all active providers to which the user is already federated.

JSP Name and Implemented APIs	Purpose
■ `TerminationDone.jsp` Invoked APIs are: ■ `LibertyManager.isTerminationSuccess(request)` ■ `LibertyManager.isTerminationCanceled(request)`	Displays the status of federation termination (success or cancelled). Status is checked using the `isTerminationCancelled(request)` method.

Using the Liberty ID-FF Federation API

The following packages form the Federation API. For more detailed information, see the *Federated Access Manager 8.0 Java API Reference*.

com.sun.identity.federation.accountmgmt

Remark 5–1 Reviewer **Public or private? Doc or no?**

Retrieves the information of federated user account

com.sun.identity.federation.common

Remark 5–2 Reviewer **Public or private? Doc or no?**

Common federation utilities

com.sun.identity.federation.message

Remark 5–3 Reviewer **Public or private? Doc or no?**

Classes for federation messages and assertions

com.sun.identity.federation.message.common

Remark 5–4 Reviewer **Public or private? Doc or no?**

common classes used by federation protocol messages

com.sun.identity.federation.plugins

The `com.sun.identity.federation.plugins` package contains the `FederationSPAdapter` interface which can be implemented to allow applications to customize their actions before and after invoking the federation protocols. For example, a service provider may want to choose to redirect to a specific location after single sign-on.

com.sun.identity.federation.services

The `com.sun.identity.federation.services` package provides interfaces for writing custom plug-ins that can be used during the federation or single sign-on process. The interfaces are described in the following table.

TABLE 5–1 `com.sun.identity.federation.services` Interfaces

Interface	Description
`FSAttributeMapper`	Plug-in for mapping the attributes passed from the identity provider to local attributes on the service provider side during the single sign-on.
`FSAttributePlugin`	Plug-in for an identity provider to add `AttributeStatements` into a SAML assertion during the single sign-on process.
`FSIDPProxy`	Interface used to find a preferred identity provider to which an authentication request can be proxied.

com.sun.liberty

The `com.sun.liberty` package contains the `LibertyManager` class which must be instantiated by web applications that want to access the Federation framework. It also contains the methods needed for account federation, session termination, log in, log out and other actions. Some of these methods are described in the following table.

TABLE 5–2 `com.sun.liberty` Methods

Method	Description
`getFederatedProviders()`	Returns a specific user's federated providers.
`getIDPFederationStatus()`	Retrieves a user's federation status with a specified identity provider. This method assumes that the user is already federated with the provider.
`getIDPList()`	Returns a list of all trusted identity providers.
`getIDPList()`	Returns a list of all trusted identity providers for the specified hosted provider.
`getProvidersToFederate()`	Returns a list of all trusted identity providers to which the specified user is not already federated.
`getSPList()`	Returns a list of all trusted service providers.
`getSPList()`	Returns a list of all trusted service providers for the specified hosted provider.
`getSPFederationStatus()`	Retrieves a user's federation status with a specified service provider. This method assumes that the user is already federated with the provider.

Executing the Federation Samples

[Remark 5–5 Reviewer: Other federation samples for WS-Federation??] need to be rewritten based on the new client WAR sample. There is no Sample1, 2, 3 anymore.

◆ ◆ ◆ CHAPTER 6

Implementing WS-Federation

Federation used to mean Liberty ID-FF, now it includes SAML 1.x and SAML2 and WS-Federation as well. Although WS-Federation and Identity Federation contains federation word, they are not related, so should separate them into two separate chapters. This chapter should be renamed as "Implementing Liberty ID-FF". The new chapter should be something like "Implementing WS-Federation".

Using the WS-Federation API

[Remark 6–1 Reviewer: Other packages used by WS-Federation??] The following packages form the WS-Federation API. For more detailed information, see the *Federated Access Manager 8.0 Java API Reference*.

`com.sun.identity.wsfederation.plugins`

Defines common WS-Federation utilities and constants.

`com.sun.identity.wsfederation.common`

Defines WS-Federation Plugin SPIs

WS-Federation Samples

Remark 6–2 Reviewer **Other federation samples for WS-Federation??**

XXXXXX

◆ ◆ ◆ CHAPTER 7

Constructing SAML Messages

Sun Java™ System Federated Access Manager has implemented two versions of the Security Assertion Markup Language (SAML). This chapter contains information on these implementations in the following sections.

SAML v2

Using the SAML v2 SDK

The SAML v2 framework provides application programming interfaces (API) that can be used to construct and process assertions, requests, and responses. The SDK is designed to be pluggable although it can also be run as a standalone application (outside of an instance of Federated Access Manager).

- For information on the packages in the SDK, see "Exploring the SAML v2 Packages" on page 128.
- For ways to set a customized implementation, see "Setting a Customized Class" on page 128.
- For instructions on how to install the SDK as a standalone application, see "Installing the SAML v2 SDK" on page 129.

Exploring the SAML v2 Packages

The SAML v2 SDK includes the following packages:

- "`com.sun.identity.saml2.assertion` Package" on page 128
- "`com.sun.identity.saml2.common` Package" on page 128
- "`com.sun.identity.saml2.protocol` Package" on page 128

For more detailed information, see the *Federated Access Manager 8.0 Java API Reference*.

`com.sun.identity.saml2.assertion` Package

This package provides interfaces to construct and process SAML v2 assertions. It also contains the `AssertionFactory`, a factory class used to obtain instances of the objects defined in the assertion schema.

`com.sun.identity.saml2.common` Package

This package provides interfaces and classes used to define common SAML v2 utilities and constants.

`com.sun.identity.saml2.protocol` Package

This package provides interfaces used to construct and process the SAML v2 request/response protocol. It also contains the `ProtocolFactory`, a factory class used to obtain object instances for concrete elements in the protocol schema.

Setting a Customized Class

There are two ways you could set a customized implementation class:

1. Add a mapping property to Federated Access Manager configuration data store in the format:

   `com.sun.identity.saml2.sdk.mapping.`*interface-name*`=`*new-class-name*

   For example, to define a customized Assertion interface, you would add:

   ```
   com.sun.identity.saml2.sdk.mapping.Assertion=
    com.ourcompany.saml2.AssertionImpl
   ```

2. Set an environment variable for the Virtual Machine for the Java™ platform (JVM™). For example, you can add the following environment variable when starting the application:

   ```
   -Dcom.sun.identity.saml2.sdk.mapping.Assertion=
    com.ourcompany.saml2.AssertionImpl
   ```

Installing the SAML v2 SDK

[Remark 7–1 Writer: "Installing the SAML v2 SDK" section need to be rewritten, user need to use our FAM client SDK based installation.] "Installing the SAML v2 SDK" section need to be rewritten, user need to use our FAM client SDK based installation.

Service Provider Interfaces

Remark 7–2 Writer **two new public API to be documented: AssertionIDRequestMapper.java SAML2ServiceProviderAdapter.java**

The `com.sun.identity.saml2.plugins` package provides pluggable interfaces to extend SAML v2 functionality into your remote application. The classes can be configured per provider entity. Default implementations are provided, but a customized implementation can be plugged in by modifying the corresponding attribute in the provider's extended metadata configuration file. The mappers include:

- "Account Mappers" on page 129
- "Attribute Mappers" on page 130
- "Authentication Context Mappers" on page 130

For more information, see the *Federated Access Manager 8.0 Java API Reference*.

Account Mappers

An account mapper is used to associate a local user account with a remote user account based on a specified attribute. A default account mapper has been developed for both sides of the SAML v2 interaction, service providers and identity providers.

- "`IDPAccountMapper`" on page 129
- "`SPAccountMapper`" on page 129

`IDPAccountMapper`

The `IDPAccountMapper` interface is used on the identity provider side to map user accounts in cases of single sign-on and federation termination. The default implementation, `com.sun.identity.saml2.plugins.DefaultIDPAccountMapper`, maps the accounts based on the persistent `NameID` attribute.

`SPAccountMapper`

The `SPAccountMapper` interface is used on the service provider side to map user accounts in cases of single sign-on and federation termination. The default implementation, `com.sun.identity.saml2.plugins.DefaultSPAccountMapper`, supports mapping based on the transient and persistent `NameID` attributes, and attribute federation based on properties defined in the extended metadata configuration file. The user mapping is based on information passed from the identity provider in an `<AttributeStatment>`.

Attribute Mappers

An attribute mapper is used to associate attribute names passed in the `<AttributeStatement>` of an assertion. A default attribute mapper has been developed for both participants in the SAML v2 interaction, service providers and identity providers. They are defined in the extended metadata configuration files and explained in the following sections:

- "`IDPAttributeMapper`" on page 130
- "`SPAttributeMapper`" on page 130
- "Setting Up Attribute Mappers" on page 130

`IDPAttributeMapper`

The `IDPAttributeMapper` interface is used by the identity provider to specify which user attributes will be included in an assertion. The default implementation, `com.sun.identity.saml2.plugins.DefaultIDPAttributeMapper`, retrieves attribute mappings (*SAML v2-attribute=user-attribute*) defined in the `attributeMap` property in the identity provider's extended metadata configuration file. It reads the value of the user attribute from the identity provider's data store, and sets this value as the `<AttributeValue>` of the specified SAML v2 attribute. The SAML v2 attributes and values are then included in the `<AttributeStatement>` of the assertion and sent to the service provider. The value of `attributeMap` can be changed to modify the mapper's behavior without programming. The default mapper itself can be modified to attach any identity provider user attribute with additional programming.

`SPAttributeMapper`

The `SPAttributeMapper` interface is used by the service provider to map attributes received in an assertion to its local attributes. The default implementation, `com.sun.identity.saml2.plugins.DefaultSPAttributeMapper`, retrieves the attribute mappings defined in the `attributeMap` property in the service provider's extended metadata configuration file. It extracts the value of the SAML v2 attribute from the assertion and returns a key/value mapping which will be set in the user's single sign-on token. The mapper can also be customized to choose user attributes from the local service provider datastore.

Setting Up Attribute Mappers

"To Setup Attribute Mapper" & "To configure mappings" sections, user need to go to console to set it instead of using saml2meta CLI.

Authentication Context Mappers

Authentication context refers to information added to an assertion regarding details of the technology used for the actual authentication action. For example, a service provider can request that an identity provider comply with a specific authentication method by identifying that method in an authentication request. The authentication context mapper pairs a standard

SAML v2 authentication context class reference (`PasswordProtectedTransport`, for example) to a Federated Access Manager authentication scheme (`module=LDAP`, for example) on the identity provider side and sets the appropriate authentication level in the user's SSO token on the service provider side. The identity provider would then deliver (with the assertion) the authentication context information in the form of an authentication context declaration added to the assertion. The process for this is described below.

1. A user accesses `spSSOInit.jsp` using the `AuthnContextClassRef` query parameter.

   For example, **`http://`***SP_host*`:`*SP_port*`/`*uri*`/spSSOInit.jsp?metaAlias=`***SP_MetaAlias***`&idpEntityID=`***IDP_EntityID***`&AuthnContextClassRef=PasswordProtec**
2. The `SPAuthnContextMapper` is invoked to map the value of the query parameter to a `<RequestedAuthnContext>` and an authentication level.
3. The service provider sends the `<AuthRequest>` with the `<RequestedAuthnContext>` to the identity provider.
4. The identity provider processes the `<AuthRequest>` by invoking the `IDPAuthnContextMapper` to map the incoming information to a defined authentication scheme.

   Note – If there is no matching authentication scheme, an authentication error page is displayed.

   ---

5. The identity provider then redirects the user (including information regarding the authentication scheme) to the Authentication Service for authentication.

   For example, **`http://`***AM_host*`:`*AM_port*`/`*uri*`/UI/Login?module=LDAP`** redirects to the LDAP authentication module.
6. After successful authentication, the user is redirected back to the identity provider for construction of a response based on the mapped authentication class reference.
7. The identity provider then returns the user to the assertion consumer on the service provider side.
8. After validating the response, the service provider creates a single sign-on token carrying the authentication level defined in the previous step.

A default authentication context mapper has been developed for both sides of the SAML v2 interaction. Details about the mappers are in the following sections:

- "`IDPAuthnContextMapper`" on page 132
- "`SPAuthnContextMapper`" on page 132

The procedure for configuring mappings is in "Configuring Mappings" on page 133.

IDPAuthnContextMapper

The IDPAuthnContextMapper is configured for the identity provider and maps incoming authentication requests from the service provider to a Federated Access Manager authentication scheme (user, role, module, level or service-based authentication), returning a response containing the authentication status to the service provider. The following attributes in the identity provider extended metadata are used by the IDPAuthnContextMapper:

- The idpAuthncontextMapper property specifies the mapper implementation.
- The idpAuthncontextClassrefMapping property specifies the mapping between a standard SAMLv2 authentication context class reference and an Access Manager authentication scheme. It takes a value in the following format:

```
authnContextClassRef | authlevel | authnType=authnValue | authnType=authnValue | ... [|default]
```

For example,
urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport|3|module=LDAP|default
maps the SAMLv2 PasswordProtectedTransport class reference to the Federated Access Manager LDAP authentication module.

SPAuthnContextMapper

The SPAuthnContextMapper is configured for the service provider and maps the parameters in incoming HTTP requests to an authentication context. It creates a <RequestedAuthnContext> element based on the query parameters and attributes configured in the extended metadata of the service provider. The <RequestedAuthnContext> element is then included in the <AuthnRequest> element sent from the service provider to the identity provider for authentication. The SPAuthnContextMapper also maps the authentication context on the identity provider side to the authentication level set as a property of the user's single sign-on token. The following query parameters can be set in the URL when accessing spSSOInit.jsp:

- AuthnContextClassRef or AuthnContextDeclRef: These properties specify one or more URI references identifying the provider's supported authentication context classes. If a value is not specified, the default is urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport.
- *AuthLevel*: This parameter specifies the authentication level of the authentication context being used for authentication.
- *AuthComparison*: This parameter specifies the method of comparison used to evaluate the requested context classes or statements. Accepted values include:
  - *exact* where the authentication context statement in the assertion must be the exact match of, at least, one of the authentication contexts specified.
  - *minimum* where the authentication context statement in the assertion must be, at least, as strong (as deemed by the identity provider) one of the authentication contexts specified.

- *maximum* where the authentication context statement in the assertion must be no stronger than any of the authentication contexts specified.
- *better* where the authentication context statement in the assertion must be stronger than any of the authentication contexts specified.

If the element is not specified, the default value is *exact*.

An example URL might be **`http://`***SP_host*`:`*SP_port*`/`*uri*`/spSSOInit.jsp?metaAlias=`***SP_MetaAlias***`&idpEntityID=`***IDP_EntityID***`&AuthnContextClassRef=PasswordProtected`**

The following attributes in the service provider extended metadata are used by the `SPAuthnContextMapper`:

- The `spAuthncontextMapper` property specifies the name of the service provider mapper implementation.
- The `spAuthncontextClassrefMapping` property specifies the map of authentication context class reference and authentication level in the following format:

  ```
  authnContextClassRef | authlevel [| default]
  ```

- The `spAuthncontextComparisonType` property is optional and specifies the method of comparison used to evaluate the requested context classes or statements. Accepted values include:
  - *exact* where the authentication context statement in the assertion must be the exact match of, at least, one of the authentication contexts specified.
  - *minimum* where the authentication context statement in the assertion must be, at least, as strong (as deemed by the identity provider) one of the authentication contexts specified.
  - *maximum* where the authentication context statement in the assertion must be no stronger than any of the authentication contexts specified.
  - *better* where the authentication context statement in the assertion must be stronger than any of the authentication contexts specified.

  If the element is not specified, the default value is *exact*.

Configuring Mappings

"To Setup Attribute Mapper" & "To configure mappings" sections, user need to go to console to set it instead of using saml2meta CLI.

Using Secure Attribute Exchange

XXXXX

JavaServer Pages

JavaServer Pages (JSP) are HTML files that contain additional code to generate dynamic content. More specifically, they contain HTML code to display static text and graphics, as well as application code to generate information. When the page is displayed in a web browser, it will contain both the static HTML content and dynamic content retrieved via the application code. The SAML v2 framework contains JSP that can initiate SAML v2 interactions. After installation, these pages can be accessed using the following URL format:

http(s)://*host*:*port*/*uri*/**saml2/jsp/***jsp-page-name***?metaAlias=***xxx***&**...

The JSP are collected in the /*path-to-context-root*/fam/saml2/config/jsp directory. The following sections contain descriptions of, and uses for, the different JSP.

Caution – The following JSP cannot be modified:

- idpArtifactResolution.jsp
- idpMNISOAP.jsp
- spMNISOAP.jsp

Default Display Page

default.jsp is the default display page for the SAML v2 framework. After a successful SAML v2 operation (single sign-on, single logout, or federation termination), a page is displayed. This page, generally the originally requested resource, is specified in the initiating request using the <RelayState> element. If a <RelayState> element is not specified, the value of the <defaultRelayState> property in the extended metadata configuration is displayed. If a <defaultRelayState> is not specified, this default.jsp is used. default.jsp can take in a message to display, for example, upon a successful authentication. The page can also be modified to add additional functionality.

Caution – When the value of <RelayState> or <defaultRelayState> contains special characters (such as &), it must be URL-encoded. For more information, see XXXXXX.

Assertion Consumer Page

The spAssertionConsumer.jsp processes the responses that a service provider receives from an identity provider. When a service provider wants to authenticate a user, it sends an authentication request to an identity provider. The AuthnRequest asks that the identity

provider return a `Response` containing one or more assertions. The `spAssertionConsumer.jsp` receives and parses the `Response` (or an artifact representing it). The endpoint for this JSP is *protocol*`://`*host*`:`*port*`/`*service-deploy-uri*`/Consumer`. Some ways in which the `spAssertionConsumer.jsp` can be customized include:

- The `localLoginUrl` parameter in the `spAssertionConsumer.jsp` retrieves the value of the `localAuthUrl` property in the service provider's extended metadata configuration. The value of `localAuthUrl` points to the local login page on the service provider side. If `localAuthUrl` is not defined, the login URL is calculated using the Assertion Consumer Service URL defined in the service provider's standard metadata configuration. Changing the `localLoginUrl` parameter value in `spAssertionConsumer.jsp` is another way to define the service provider's local login URL.
- After a successful single sign-on and before the final protected resource (defined in the `<RelayState>` element) is accessed, the user may be directed to an intermediate URL, if one is configured as the value of the `intermediateUrl` property in the service provider's extended metadata configuration file. For example, this intermediate URL might be a successful account creation page after the auto-creation of a user account. The `redirectUrl` in `spAssertionConsumer.jsp` can be modified to override the `intermediateUrl` value.

Single Sign-on Pages

The single sign-on JSP are used to initiate single sign-on and, parse authentication requests, and generate responses. These include:

- "`idpSSOFederate.jsp`" on page 135
- "`idpSSOInit.jsp`" on page 135
- "`spSSOInit.jsp`" on page 136

`idpSSOFederate.jsp`

`idpSSOFederate.jsp` works on the identity provider side to receive and parse authentication requests from the service provider and generate a `Response` containing an assertion. The endpoint for this JSP is *protocol*`://`*host*`:`*port*`/`*service-deploy-uri*`/idpSSOFederate`. `idpSSOFederate.jsp` takes the following parameters:

- `SAMLRequest`: This required parameter takes as a value the XML blob that contains the `AuthnRequest`.
- `metaAlias`: This optional parameter takes as a value the `metaAlias` set in the identity provider's extended metadata configuration file.
- `RelayState`: This optional parameter takes as a value the target URL of the request.

`idpSSOInit.jsp`

`idpSSOInit.jsp` initiates single sign-on from the identity provider side (also referred to as *unsolicited response*). For example, a user requests access to a resource. On receiving this

request for access, `idpSSOInit.jsp` looks for a cached assertion which, if present, is sent to the service provider in an unsolicited `<Response>`. If no assertion is found, `idpSSOInit.jsp` verifies that the following required parameters are defined:

- `metaAlias`: This parameter takes as a value the `metaAlias` set in the identity provider's extended metadata configuration file. If the `metaAlias` attribute is not present, an error is returned.
- `spEntityID`: The entity identifier of the service provider to which the response is sent.

If defined, the unsolicited `Response` is created and sent to the service provider. If not, an error is returned. The endpoint for this JSP is *protocol*`://`*host*`:`*port*`/`*service-deploy-uri*`/idpssoinit`. The following optional parameters can also be passed to `idpSSOInit.jsp`:

- `RelayState`: The target URL of the request.
- `NameIDFormat`: The currently supported name identifier formats: *persistent* or *transient*.
- `binding`: A URI suffix identifying the protocol binding to use when sending the `Response`. The supported values are:
  - `HTTP-Artifact`
  - `HTTP-POST`

spSSOInit.jsp

`spSSOInit.jsp` is used to initiate single sign-on from the service provider side. On receiving a request for access, `spSSOInit.jsp` verifies that the following required parameters are defined:

- `metaAlias`: This parameter takes as a value the `metaAlias` set in the identity provider's extended metadata configuration file. If the `metaAlias` attribute is not present, an error is returned.
- `idpEntityID`: The entity identifier of the identity provider to which the request is sent. If `idpEntityID` is not provided, the request is redirected to the SAML v2 IDP Discovery Service to get the user's preferred identity provider. In the event that more then one identity provider is returned, the last one in the list is chosen. If `idpEntityID` cannot be retrieved using either of these methods, an error is returned.

If defined, the `Request` is created and sent to the identity provider. If not, an error is returned. The endpoint for this JSP is *protocol*`://`*host*`:`*port*`/`*service-deploy-uri*`/spssoinit`. The following optional parameters can also be passed to `spSSOInit.jsp`:

- `RelayState`: The target URL of the request.
- `NameIDFormat`: The currently supported name identifier formats: *persistent* or *transient*.
- `binding`: A URI suffix identifying the protocol binding to use when sending the `Response`. The supported values are:
  - `HTTP-Artifact`
  - `HTTP-POST`

- `AssertionConsumerServiceIndex`: An integer identifying the location to which the `Response` message should be returned to the requester. requester. It applies to profiles in which the requester is different from the presenter, such as the Web Browser SSO profile.
- `AttributeConsumingServiceIndex`: An integer indirectly specifying information (associated with the requester) describing the SAML attributes the requester desires or requires to be supplied.
- `isPassive`: Takes a value of `true` or `false` with `true` indicating the identity provider should authenticate passively.
- `ForceAuthN`: Takes a value of `true` indicating that the identity provider must force authentication or `false` indicating that the identity provider can reuse existing security contexts.
- `AllowCreate`: Takes a value of `true` indicating that the identity provider is allowed to created a new identifier for the principal if it does not exist or `false`.
- `Destination`: A URI indicating the address to which the request has been sent.
- `AuthnContextClassRef`: Specifies a URI reference identifying an authentication context class that describes the declaration that follows. Multiple references can be pipe-separated.
- `AuthnContextDeclRef`: Specifies a URI reference to an authentication context declaration. Multiple references can be pipe-separated.
- `AuthComparison`: The comparison method used to evaluate the requested context classes or statements. Accepted values include: *minimum*, *maximum* or *better*.
- `Consent`: Indicates whether or not (and under what conditions) consent has been obtained from a principal in the sending of this request.

Note – `Consent` is not supported in this release.

Name Identifier Pages

The various *ManageNameID* (MNI) JSP provide a way to change account identifiers or terminate mappings between identity provider accounts and service provider accounts. For example, after establishing a name identifier for use when referring to a principal, the identity provider may want to change its value and/or format. Additionally, an identity provider might want to indicate that a name identifier will no longer be used to refer to the principal. The identity provider will notify service providers of the change by sending them a `ManageNameIDRequest`. A service provider also uses this message type to register or change the `SPProvidedID` value (included when the underlying name identifier is used to communicate with it) or to terminate the use of a name identifier between itself and the identity provider.

- "`idpMNIRequestInit.jsp`" on page 138
- "`idpMNIRedirect.jsp`" on page 138
- "`spMNIRequestInit.jsp`" on page 138
- "`spMNIRedirect.jsp`" on page 139

idpMNIRequestInit.jsp

`idpMNIRequestInit.jsp` initiates the `ManageNameIDRequest` at the identity provider by user request. The endpoint for this JSP is *protocol*`://`*host*`:`*port*/*service-deploy-uri*`/IDPMniInit`. It takes the following required parameters:

- `metaAlias`: The value of the `metaAlias` property set in the identity provider's extended metadata configuration file. If the `metaAlias` attribute is not present, an error is returned.
- `spEntityID`: The entity identifier of the service provider to which the response is sent.
- `requestType`: The type of `ManageNameIDRequest`. Accepted values include `Terminate` and `NewID`.

Note – `NewID` is not supported in this release.

Some of the other optional parameters are :

- `binding`: A URI specifying the protocol binding to use for the `<Request>`. The supported values are:
  - `urn:oasis:names:tc:SAML:2.0:bindings:SOAP`
  - `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect`
- `RelayState`: The target URL of the request

idpMNIRedirect.jsp

`idpMNIRedirect.jsp` processes the `ManageNameIDRequest` and the `ManageNameIDResponse` received from the service provider using HTTP-Redirect. The endpoint for this JSP is *protocol*`://`*host*`:`*port*/*service-deploy-uri*`/IDPMniRedirect`. It takes the following required parameters:

- `SAMLRequest`: The `ManageNameIDRequest` from the service provider.
- `SAMLResponse`: The `ManageNameIDResponse` from the service provider.

Optionally, it can also take the `RelayState` parameter which specifies the target URL of the request.

spMNIRequestInit.jsp

`spMNIRequestInit.jsp` initiates the `ManageNameIDRequest` at the service provider by user request. The endpoint for this JSP is *protocol*`://`*host*`:`*port*/*service-deploy-uri*`/SPMniInit`. It takes the following required parameters:

- `metaAlias`: This parameter takes as a value the `metaAlias` set in the identity provider's extended metadata configuration file. If the `metaAlias` attribute is not present, an error is returned.

- `idpEntityID`: The entity identifier of the identity provider to which the request is sent.
- `requestType`: The type of `ManageNameIDRequest`. Accepted values include `Terminate` and `NewID`.

Note – `NewID` is not supported in this release.

Some of the other optional parameters are :

- `binding`: A URI specifying the protocol binding to use for the `Request`. The supported values are:
  - `urn:oasis:names:tc:SAML:2.0:bindings:SOAP`
  - `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect`
- `RelayState`: The target URL of the request.

spMNIRedirect.jsp

`spMNIRedirect.jsp` processes the `ManageNameIDRequest` and the `<ManageNameIDResponse>` received from the identity provider using HTTP-Redirect. The endpoint for this JSP is *protocol*`://`*host*`:`*port*/*service-deploy-uri*`/SPMniRedirect`. It takes the following required parameters:

- `SAMLRequest`: The `ManageNameIDRequest` from the identity provider.
- `SAMLResponse`: The `ManageNameIDResponse` from the identity provider.

Optionally, it can also take the `RelayState` parameter which specifies the target URL of the request.

Single Logout JavaServer Pages

The single logout JSP provides the means by which all sessions authenticated by a particular identity provider are near-simultaneously terminated. The single logout protocol is used either when a user logs out from a participant service provider or when the principal logs out directly from the identity provider.

- "`idpSingleLogoutInit.jsp`" on page 139
- "`idpSingleLogoutRedirect.jsp`" on page 140
- "`spSingleLogoutInit.jsp`" on page 140
- "`spSingleLogoutRedirect.jsp`" on page 141

idpSingleLogoutInit.jsp

`idpSingleLogoutInit.jsp` initiates a `LogoutRequest` at the identity provider by user request. The endpoint for this JSP is *protocol*`://`*host*`:`*port*/*service-deploy-uri*`/IDPSloInit`. There are no required parameters. Optional parameters include:

- `RelayState`: The target URL after single logout.
- `binding`: A URI specifying the protocol binding to use for the `<Request>`. The supported values are:
  - `urn:oasis:names:tc:SAML:2.0:bindings:SOAP`
  - `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect`
- `Destination`: A URI indicating the address to which the request has been sent.
- `Consent`: Indicates whether or not (and under what conditions) consent has been obtained from a principal in the sending of this request.

  Note – `Consent` is not supported in this release.

- `Extension`: Specifies permitted extensions as a list of string objects.

  Note – `Extension` is not supported in this release.

- `logoutAll`: Specifies that the identity provider send log out requests to all service providers without a session index. It will logout all sessions belonging to the user.

`idpSingleLogoutRedirect.jsp`

`idpSingleLogoutRedirect.jsp` processes the `LogoutRequest` and the `LogoutResponse` received from the service provider using HTTP-Redirect. The endpoint for this JSP is *protocol*`://`*host*`:`*port*`/`*service-deploy-uri*`/IDPSloRedirect`. It takes the following required parameters:

- `SAMLRequest`: The `LogoutRequest` from the service provider.
- `SAMLResponse`: The `LogoutResponse` from the service provider.

Optionally, it can also take the `RelayState` parameter which specifies the target URL of the request.

`spSingleLogoutInit.jsp`

`spSingleLogoutInit.jsp` initiates a `LogoutRequest` at the identity provider by user request. The endpoint for this JSP is *protocol*`://`*host*`:`*port*`/`*service-deploy-uri*`/SPSloInit`. There are no required parameters. Optional parameters include:

- `RelayState`: The target URL after single logout.
- `binding`: A URI specifying the protocol binding to use for the `<Request>`. The supported values are:
  - `urn:oasis:names:tc:SAML:2.0:bindings:SOAP`

- `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect`

- `Destination`: A URI indicating the address to which the request has been sent.
- `Consent`: Indicates whether or not (and under what conditions) consent has been obtained from a principal in the sending of this request.

  Note – `Consent` is not supported in this release.

- `Extension`: Specifies permitted extensions as a list of string objects.

  Note – `Extension` is not supported in this release.

`spSingleLogoutRedirect.jsp`

`spSingleLogoutRedirect.jsp` processes the `LogoutRequest` and the `LogoutResponse` received from the identity provider using HTTP-Redirect. The endpoint for this JSP is *protocol*`://`*host*`:`*port*/*service-deploy-uri*`/SPSloRedirect`. It takes the following required parameters:

- `SAMLRequest`: The `LogoutRequest` from the identity provider.
- `SAMLResponse`: The `LogoutResponse` from the identity provider.

Optionally, it can also take the `RelayState` parameter which specifies the target URL of the request.

SAML v2 Samples

Need info on SAML2 samples

SAML 1.x

The following sections contain information on the SAML 1.x framework.

Interfaces

Federated Access Manager contains a SAML 1.x API that consists of several Java packages. Administrators can use these packages to integrate the SAML functionality and XML messages

into their applications and services. The API supports all types of assertions and operates with the Federated Access Manager authorities to process external SAML 1.x requests and generate SAML 1.x responses. The packages include the following:

For more detailed information, including methods and their syntax and parameters, see the *Federated Access Manager 8.0 Java API Reference*.

`com.sun.identity.saml` Package

This package contains the following classes.

`AssertionManager` Class

The `AssertionManager` class provides interfaces and methods to create and get assertions, authentication assertions, and assertion artifacts. This class is the connection between the SAML specification and Federated Access Manager. Some of the methods include the following:

- `createAssertion` creates an assertion with an authentication statement based on an Federated Access Manager SSO Token ID.
- `createAssertionArtifact` creates an artifact that references an assertion based on an Federated Access Manager SSO Token ID.
- `getAssertion` returns an assertion based on the given parameter (given artifact, assertion ID, or query).

`SAMLClient` Class

The `SAMLClient` class provides methods to execute either the Web Browser Artifact Profile or the Web Browser POST Profile from within an application as opposed to a web browser. Its methods include the following:

- `getAssertionByArtifact` returns an assertion for a corresponding artifact.
- `doWebPOST` executes the Web Browser POST Profile.
- `doWebArtifact` executes the Web Browser Artifact Profile.

`com.sun.identity.saml.assertion` Package

This package contains the classes needed to create, manage, and integrate an XML assertion into an application. The following code example illustrates how to use the `Attribute` class and `getAttributeValue` method to retrieve the value of an attribute. From an assertion, call the `getStatement()` method to retrieve a set of statements. If a statement is an attribute statement, call the `getAttribute()` method to get a list of attributes. From there, call `getAttributeValue()` to retrieve the attribute value.

EXAMPLE 7–1 Sample Code to Obtain an Attribute Value

```
// get statement in the assertion
Set set = assertion.getStatement();
//assume there is one AttributeStatement
//should check null& instanceof
AttributeStatement statement = (AttributeStatement) set.iterator().next();
List attributes = statement.getAttribute();
// assume there is at least one Attribute
Attribute attribute = (Attribute) attributes.get(0);
List values = attribute.getAttributeValue();
```

`com.sun.identity.saml.common` Package

This package defines classes common to all SAML elements, including site ID, issuer name, and server host. The package also contains all SAML-related exceptions.

`com.sun.identity.saml.plugins` Package

The SAML 1.x framework provides service provider interfaces (SPIs), three of which have default implementations. The default implementations of these SPIs can be altered, or brand new ones written, based on the specifications of a particular customized service. The implementations are then used to integrate SAML into the custom service. Currently, the package includes the following.

- "`ActionMapper` Interface" on page 143
- "`AttributeMapper` Interface" on page 144
- "`NameIdentifierMapper` Interface" on page 144
- "`PartnerAccountMapper` Interface" on page 144
- "`PartnerSiteAttributeMapper` Interface" on page 144

`ActionMapper` Interface

`ActionMapper` is an interface used to obtain single sign-on information and to map partner actions to Federated Access Manager authorization decisions. A default action mapper is provided if no other implementation is defined.

`AttributeMapper` Interface

`AttributeMapper` is an interface used in conjunction with an `AttributeQuery` class When a site receives an attribute query, this mapper obtains the `SSOToken` or an assertion (containing an authentication statement) from the query. The retrieved information is used to convert the attributes in the query to the corresponding Federated Access Manager attributes. A default attribute mapper is provided if no other implementation is defined.

For more information, see XXXXX.

`NameIdentifierMapper` Interface

`NameIdentifierMapper` is an interface that can be implemented by a site to map a user account to a name identifier in the subject of a SAML assertion. The implementation class is specified when configuring the site's Trusted Partners.

`PartnerAccountMapper` Interface

Remark 7–3 Reviewer **New**

Caution – The `AccountMapper` interface has been deprecated. Use the `PartnerAccountMapper` interface.

The `PartnerAccountMapper` interface needs to be implemented by each partner site. The implemented class maps the partner site's user accounts to user accounts configured in Access Manager for purposes of single sign-on. For example, if single sign-on is configured from site A to site B, a site-specific account mapper can be developed and defined in the Trusted Partners sub-attribute of site B's Trusted Partners profile. When site B processes the assertion received, it locates the corresponding account mapper by retrieving the source ID of the originating site. The `PartnerAccountMapper` takes the whole assertion as a parameter, enabling the partner to define user account mapping based on attributes inside the assertion. The default implementation is `com.sun.identity.saml.plugin.DefaultAccountMapper`. If a site-specific account mapper is not configured, this default mapper is used.

Note – Turning on the Debug Service in the Federated Access Manager configuration data store logs additional information about the account mapper, for example, the user name and organization to which the mapper has been mapped.

`PartnerSiteAttributeMapper` Interface

Remark 7–4 Reviewer **New**

Caution – The `SiteAttributeMapper` interface has been deprecated. Use the `PartnerSiteAttributeMapper` interface.

The `PartnerSiteAttributeMapper` interface needs to be implemented by each partner site. The implemented class defines a list of attributes to be returned as elements of the `AttributeStatements` in an authentication assertion. By default, when Federated Access Manager creates an assertion and no mapper is specified, the authentication assertion only contains authentication statements. If a partner site wants to include attribute statements, it needs to implement this mapper which would be used to obtain attributes, create the attribute statement, and insert the statement inside the assertion. To set up a `PartnerSiteAttributeMapper` do the following:

1. Implement a customized class based on the `PartnerSiteAttributeMapper` interface.

   This class will include user attributes in the SAML authentication assertion.
2. Log in to the Federated Access Manager console to configure the class in the Site Attribute Mapper attribute of the Trusted Partner configuration.

   See XXXXX for more information.

`com.sun.identity.saml.protocol` Package

This package contains classes that parse the request and response XML messages used to exchange assertions and their authentication, attribute, or authorization information.

`AuthenticationQuery` Class

The `AuthenticationQuery` class represents a query for an authentication assertion. When an identity attempts to access a trusted partner web site, a SAML 1.x request with an `AuthenticationQuery` inside is directed to the authority site.

The Subject of the `AuthenticationQuery` must contain a `SubjectConfirmation` element. In this element, `ConfirmationMethod` needs to be set to `urn:com:sun:identity`, and `SubjectConfirmationData` needs to be set to the `SSOToken` ID of the Subject. If the Subject contains a `NameIdentifier`, the value of the `NameIdentifier` should be the same as the one in the `SSOToken`.

`AttributeQuery` Class

The `AttributeQuery` class represents a query for an identity's attributes. When an identity attempts to access a trusted partner web site, a SAML 1.x request with an `AttributeQuery` is directed to the authority site.

You can develop an attribute mapper to obtain an SSOToken, or an assertion that contains an AuthenticationStatement from the query. If no attribute mapper for the querying site is defined, the DefaultAttributeMapper will be used. To use the DefaultAttributeMapper, the query should have either the SSOToken or an assertion that contains an AuthenticationStatement in the SubjectConfirmationData element. If an SSOToken is used, the ConfirmationMethod must be set to urn:com:sun:identity:. If an assertion is used, the assertion should be issued by the Federated Access Manager instance processing the query or a server that is trusted by the Federated Access Manager instance processing the query.

Note – In the DefaultAttributeMapper, a subject's attributes can be queried using another subject's SSOToken if the SSOToken has the privilege to retrieve the attributes.

For a query using the DefaultAttributeMapper, any matching attributes found will be returned. If no AttributeDesignator is specified in the AttributeQuery, all attributes from the services defined under the userServiceNameList in amSAML.properties will be returned. The value of the userServiceNameList property is user service names separated by a comma.

AuthorizationDecisionQuery Class

The AuthorizationDecisionQuery class represents a query about a principal's authority to access protected resources. When an identity attempts to access a trusted partner web site, a SAML request with an AuthorizationDecisionQuery is directed to the authority site.

You can develop an ActionMapper to obtain the SSOToken ID and retrieve the authentication decisions for the actions defined in the query. If no ActionMapper for the querying site is defined, the DefaultActionMapper will be used. To use the DefaultActionMapper, the query should have the SSOToken ID in the SubjectConfirmationData element of the Subject. If the SSOToken ID is used, the ConfirmationMethod must be set to urn:com:sun:identity:. If a NameIdentifier is present, the information in the SSOToken must be the same as the information in the NameIdentifier.

Note – When using web agents, the DefaultActionMapper handles actions in the namespace urn:oasis:names:tc:SAML:1.0:ghpp only. Web agents serve the policy decisions for this action namespace.

The authentication information can also be passed through the Evidence element in the query. Evidence can contain an AssertionIDReference, an assertion containing an AuthenticationStatement issued by the Federated Access Manager instance processing the query, or an assertion issued by a server that is trusted by the Federated Access Manager instance processing the query. The Subject in the AuthenticationStatement of the Evidence element should be the same as the one in the query.

Note – Policy conditions can be passed through `AttributeStatements` of assertion(s) inside the `Evidence` of a query. If the value of an attribute contains a TEXT node only, the condition is set as `attributeName=attributeValueString`. Otherwise, the condition is set as `attributename=attributeValueElement`.

The following example illustrates one of many ways to form an authorization decision query that will return a decision.

EXAMPLE 7–2 `AuthorizationDecisionQuery` Code Sample

```
// testing getAssertion(authZQuery): no SC, with ni, with
// evidence(AssertionIDRef, authN, for this ni):
    String nameQualifier = "dc=iplanet,dc=com";
    String pName = "uid=amadmin,ou=people,dc=iplanet,dc=com";
    NameIdentifier ni = new NameIdentifier(pName, nameQualifier);
    Subject subject = new Subject(ni);
    String actionNamespace = "urn:test";
    // policy should be added to this resource with these
    // actions for the subject
    Action action1 = new Action(actionNamespace, "GET");
    Action action2 = new Action(actionNamespace, "POST");
    List actions = new ArrayList();
    actions.add(action1);
    actions.add(action2);
    String resource = "http://www.sun.com:80";
    eviSet = new HashSet();
    // this assertion should contain authentication assertion for
    // this subject and should be created by a trusted server
    eviSet.add(eviAssertionIDRef3);
    evidence = new Evidence(eviSet);
    authzQuery = new AuthorizationDecisionQuery(eviSubject1, actions,
                        evidence, resource);
    try {
        assertion = am.getAssertion(authzQuery, destID);
    } catch (SAMLException e) {
        out.println("--failed. Exception:" + e);
    }
```

`com.sun.identity.saml.xmlsig` Package

All SAML 1.x assertions, requests, and responses can be signed using this signature package. It contains SPI that are implemented to plug in proprietary XML signatures. This package contains classes needed to sign and verify using XML signatures. By default, the keystore provided with the Java Development Kit is used and the key type is DSA. The configuration properties for this functionality are in the Federated Access Manager configuration data store. For details on how to use the signature functionality, see "SAML 1.x Samples" on page 148.

SAML 1.x Samples

Remark 7–5 Writer **As of 3/17/08 no saml1x samples integrated in FAM8.**

You can access several SAML-based samples from the "SAML 1.x Samples" on page 148 installation in */path-to-context-root*`/fam/samples/saml`. These samples illustrate how the SAML 1.x framework can be used in different ways, including the following:

- A sample that serves as the basis for using the SAML client API. This sample is located in */path-to-context-root*`/fam/samples/saml/client`.
- A sample that illustrates how to form a Query, write an `AttributeMapper`, and send and process a SOAP message using the SAML SDK. This sample is located in */path-to-context-root*`/fam/samples/saml/query`.
- A sample application for achieving SSO using either the Web Browser Artifact Profile or the Web Browser POST Profile. This sample is located in */path-to-context-root*`/fam/samples/saml/sso`.
- A sample that illustrates how to use the XMLSIG API and explains how to configure for XML signing. This sample is located in */path-to-context-root*`/fam/samples/saml/xmlsig`.

Each sample includes a README file with information and instructions on how to use it.

◆ ◆ ◆ CHAPTER 8

Implementing Web Services

Federated Access Manager contains web services that can be used to extend the functionality of your federated environment. Additionally, new web services can be developed. This chapter covers the following topics:

Developing New Web Services

Any web service that is plugged into the Federated Access Manager Liberty ID-WSF framework must register a *key*, and an implementation of the `com.sun.identity.liberty.ws.soapbinding.RequestHandler` interface, with the SOAP Binding Service. (For example, the Liberty Personal Profile Service is registered with the key `idpp`, and the class `com.sun.identity.liberty. ws.soapbinding.PPRequestHandler`.) The Key value becomes part of the URL for the web service's endpoint (as in *protocol*`://`*host*`:`*port*/*deploymenturi*`/Liberty/`*key*). The implemented class allows the web service to retrieve the request (containing the authenticated principal and the authenticated security mechanism along with the entire SOAP message). The web service processes the request and generates a response. This section contains the process you would use to add a new Liberty ID-WSF web service to the Federated Access Manager framework. Instructions for some of these steps are beyond the scope of this guide. The process has been divided into two tasks:

- "To Host a Custom Service" on page 150
- "To Invoke the Custom Service" on page 157

▼ To Host a Custom Service

Before You Begin The XML Schema Definition (XSD) file written to define the new service is the starting point for developing the service's server-side code. More information can be found in XXXXXXSchema Files and Service Definition Documents.

1 Write an XML service schema for the new web service and Java classes to parse and process the XML messages.

The following sample schema defines a stock quote web service. The `QuoteRequest` and `QuoteResponse` elements define the parameters for the request and response that are inserted in the SOAP Body of the request and response, respectively. You will need to have `QuoteRequest.java` and `QuoteResponse.java` to parse and process the XML messages.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="urn:com:sun:liberty:sample:stockticker"
  targetNamespace="urn:com:sun:liberty:sample:stockticker">
  <xs:annotation>
      <xs:documentation>
         This is a sample stock ticker web service protocol
      </xs:documentation>
  </xs:annotation>

  <xs:element name="QuoteRequest" type="QuoteRequestType"/>
  <xs:complexType name="QuoteRequestType">
    <xs:sequence>
        <xs:element name = "ResourceID" type="xs:string" minOccurs="0"/>
        <xs:element name = "Symbol" type="xs:string" minOccours="1"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="PriceType">
      <xs:sequence>
          <xs:element name="Last" type="xs:integer"/>
          <xs:element name="Open" type="xs:integer"/>
          <xs:element name="DayRange" type="xs:string"/>
          <xs:element name="Change" type="xs:string"/>
          <xs:element name="PrevClose" type="xs:integer"/>
      </xs:sequence>
  </xs:complexType>

  <xs:element name="QuoteResponse" type="QuoteResponseType"/>
  <xs:complexType name="QuoteResponseType">
```

```
    <xs:sequence>
        <xs:element name="Symbol" type="xs:string"/>
        <xs:element name="Time" type="xs:dateTime"/>
        <xs:element name="Delay" type="xs:dateTime" minOccurs="0"/>
        <xs:element name="Price" type="PriceType"/>
        <xs:element name="Volume" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

2 Provide an implementation for one of the following interfaces based on the type of web service being developed:

- `com.sun.identity.liberty.ws.soapbinding.RequestHandler` for developing and deploying a general web service.

  See XXXXX.

- `com.sun.identity.liberty.ws.dst.service.DSTRequestHandler` for developing and deploying an identity data service type web service based on the Liberty Alliance Project Identity Service Interface Specifications (Liberty ID-SIS).

  See XXXXX.

In Federated Access Manager, each web service must implement one of these interfaces to accept incoming message requests and return outgoing message responses. The following sample implements the `com.sun.identity.liberty.ws.soapbinding.RequestHandler` interface for the stock quote web service.
`com.sun.identity.liberty.ws.soapbinding.Message` is the API used to construct requests and responses.

```
public class StockTickerService implements RequestHandler {
        :
         //implement business logic

         public Message processRequest(Message msg) throws
                        SOAPFaultException, Exception {
                :
                SSOToken token = (SSOToken)msg.getToken();
                List responseBody = processSOAPBody(msg.getBodies());
                :
                Message response = new Message();
                response.setBodies(responseBody);

                return response;
          }
          :
          //more business logic
```

```
}
```

3 **Compile the Java source code.**

Be sure to include `am_services.jar` in your classpath.

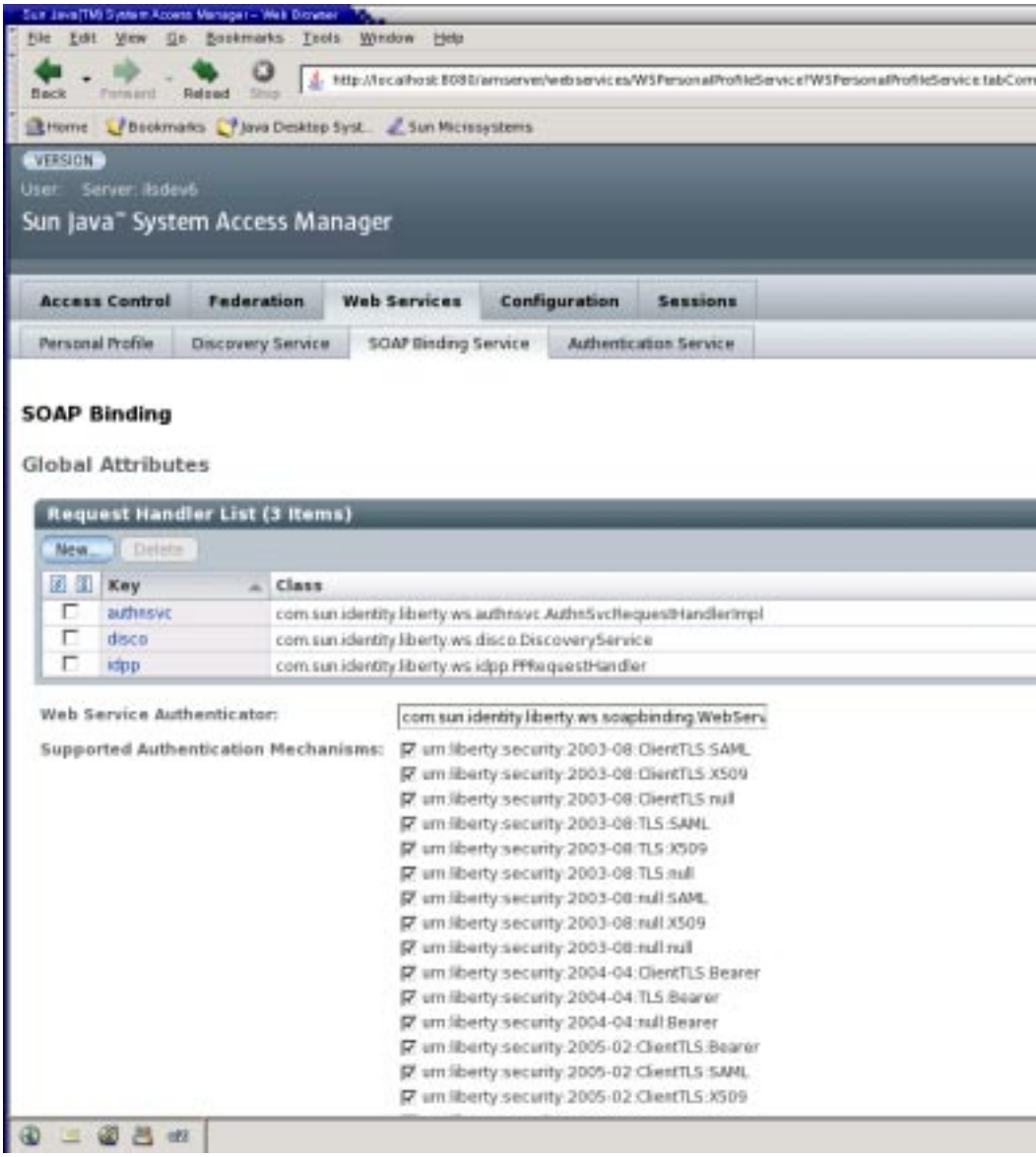
4 **Add the previously created classes to the web container classpath and restart the web container.**

5 **Login to the Federated Access Manager console as the top level administrator.**

By default, `amadmin`.

6 **Click the Web Services tab.**

7 **Under Web Services, click the SOAP Binding Service tab to register the new implementation with the SOAP Binding Service.**



8 Click New under the Request Handler List global attribute.

9 Enter a name for the implementation in the Key field.

This value will be used as part of the service endpoint URL for the web service. For example, if the value is *stock*, the endpoint URL to access the stock quote web service will be:

`http://`*SERVER_HOST*`:`*SERVER_PORT*`/`*SERVER_DEPLOY_URI*`/Liberty/stock`

10 **Enter the name of the implementation class previously created in the Class field.**

11 **(Optional) Enter a SOAP Action in the SOAP Action field.**

12 **Click Save to save the configuration.**

The request handler will be displayed under the Request Handler List.

13 **Click on the Access Control tab to begin the process of publishing the web service to the Discovery Service.**

The Discovery Service is a registry of web services. It matches the properties in a request with the properties in its registry and returns the appropriate service location. See XXXXX.

14 **Select the name of the realm to which you want to add the web service.**

15 **Select Services to access the realm's services.**

16 **Click Discovery Service.**

If the Discovery Service has not yet been added, do the following.

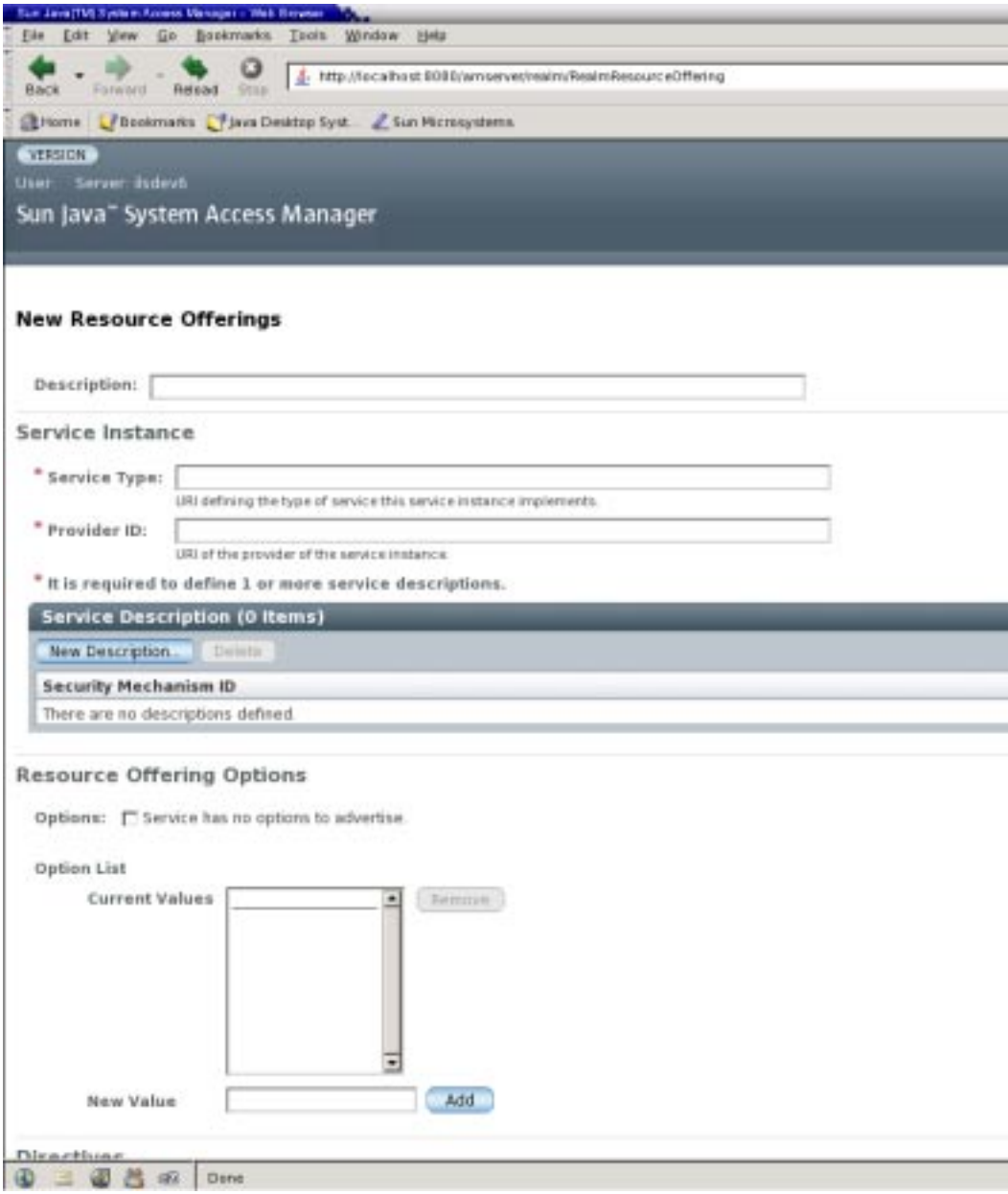
a. **Click Add.**

A list of available services is displayed.

b. **Select Discovery Service and click Next to add the service.**

The list of added services is displayed including the Discovery Service.

17 **Click Add to create a new resource offering.**



18 (Optional) Enter a description of the resource offering in the Description field.

19 Type a URI for the value of the Service Type attribute.

This URI defines the type of service. It is *recommended* that the value of this attribute be the `targetNamespace` URI defined in the *abstract* WSDL description for the service. An example of a valid URI for the sample service is `urn:com:sun:liberty:sample:stockticker`.

20 Type a URI for the value of the Provider ID attribute.

The value of this attribute contains the URI of the provider of the service instance. This information is useful for resolving trust metadata needed to invoke the service instance. A single physical provider may have multiple provider IDs.

Note – The provider represented by the URI in the Provider ID attribute must also have an entry in the `ResourceIDMapper` attribute. For more information, see XXXXXClasses For ResourceIDMapper Plug-in.

---

21 Click New Description to define the Service Description.

For each resource offering, at least one service description must be created.

a. Select the values for the Security Mechanism ID attribute to define how a web service client can authenticate to a web service provider.

This field lists the security mechanisms that the service instance supports. Select the security mechanisms that you want to add and click Add. To prioritize the list, select the mechanism and click Move Up or Move Down.

b. Type a value for the End Point URL.

This value is the URL to access the new web service. For this example, it should be:

`http://`*SERVER_HOST*`:`*SERVER_PORT*`/`*SERVER_DEPLOY_URI*`/Liberty/stock`

c. (Optional) Type a value for the SOAP Action.

This value is the equivalent of the `wsdlsoap:soapAction` attribute of the `wsdlsoap:operation` element in the service's concrete WSDL-based description.

d. Click OK to complete the configuration.

22 Check the Options box if there are no options or add a URI to specify options for the resource offering.

This field lists the options that are available for the resource offering. Options provide hints to a potential requestor about the availability of certain data or operations to a particular offering. The set of possible URIs are defined by the service type, not the Discovery Service. If no option is specified, the service instance does not display any available options. For a standard set of options, see the *Liberty ID-SIS Personal Profile Service Specification*.

23 Select a directive for the resource offering.

Directives are special entries defined in SOAP headers that can be used to enforce policy-related decisions. You can choose from the following:

- `GenerateBearerToken` specifies that a bearer token be generated.
- `AuthenticateRequester` must be used with any service description that use SAML for message authentication.
- `EncryptResourceID` specifies that the Discovery Service encrypt the resource ID.
- `AuthenticateSessionContext` is specified when a Discovery Service provider includes a SAML assertion containing a `SessionContextStatement` in any future `QueryResponse` messages.
- `AuthorizeRequester` is specified when a Discovery Service provider wants to include a SAML assertion containing a `ResourceAccessStatement` in any future `QueryResponse` messages.

If you want to associate a directive with one or more service descriptions, select the check box for that Description ID. If no service descriptions are selected, the directive is applied to all description elements in the resource offering.

24 Click OK.

25 Logout from the console.

▼ To Invoke the Custom Service

Web service clients can access the custom web service by discovering the web service's end point and using the required credentials. This information is stored by the Federated Access Manager Discovery Service. There are two ways in which a client can authenticate to Federated Access Manager in order to access the Discovery Service:

- The Liberty ID-FF is generally used if it's a browser-based application and the web service client is a federation enabled service provider.
- The Access Manager Authentication Web Service (based on the Liberty ID-WSF) is used for remote web services clients with pure SOAP-based authentication capabilities.

In the following procedure, we use the Liberty ID-WSF client API to invoke the web service.

Note – The code in this procedure is used to demonstrate the usage of the Liberty ID-WSF client API. More information can be found in the *Federated Access Manager 8.0 Java API Reference*.

---

1 Write code to authenticate the WSC to the Authentication Web Service of Federated Access Manager.

The sample code below will allow access to the Discovery Service. It is a client-side program to be run inside the WSC application.

```
public class StockClient {
             :
    public SASLResponse authenticate(
        String userName,
        String password,
        String authurl) throws Exception {

        SASLRequest saslReq =
                       new SASLRequest(AuthnSvcConstants.MECHANISM_PLAIN);
        saslReq.setAuthzID(userName);

        SASLResponse saslResp = AuthnSvcClient.sendRequest(saslReq, authurl);
        String statusCode = saslResp.getStatusCode();
        if (!statusCode.equals(SASLResponse.CONTINUE)) {
                return null;
        }

        String serverMechanism = saslResp.getServerMechanism();
        saslReq  = new SASLRequest(serverMechanism);
        String dataStr = userName + "\0" + userName + "\0" + password;
        saslReq.setData(dataStr.getBytes("UTF-8"));
        saslReq.setRefToMessageID(saslResp.getMessageID());
        saslResp = AuthnSvcClient.sendRequest(saslReq, authurl);
        statusCode = saslResp.getStatusCode();
        if (!statusCode.equals(SASLResponse.OK)) {
            return null;
        }

        return saslResp;
    }
                    :

           }
```

2 Add code that will extract the Discovery Service information from the Authentication Response.

The following additional code would be added to what was developed in the previous step.

```
ResourceOffering discoro = saslResp.getResourceOffering();
          List credentials = authnResponse.getCredentials();
```

3 Add code to query the Discovery Service for the web service's resource offering by using the Discovery Service resource offering and the credentials that are required to access it.

The following additional code would be added to what was previously developed.

```
RequestedService rs = new RequestedService(null,
                  "urn:com:sun:liberty:sample:stockticker");
              List rss = new ArrayList();
              rss.add(rs);

             Query  discoQuery = new Query(discoro.getResourceID(), rss);

             DiscoveryClient discoClient = null;

             discoClient = new DiscoveryClient(secAssertion, serviceURL, null);

             QueryResponse queryResponse = discoClient.getResourceOffering(discoQuery);
```

4 The discovery response contains the service's resource offering and the credentials required to access the service.

`quotes` contains the response body (the stock quote). You would use the Federated Access Manager SOAP API to get the body elements.

```
 List offerings = discoResponse.getResourceOffering();
          ResourceOffering stockro = (ResourceOffering)offerings.get(0);

          List credentials = discoResponse.getCredentials();

          SecurityAssertion secAssertion = null;
          if(credentials != null && !credentials.isEmpty()) {
             secAssertion = (SecurityAssertion)credentials.get(0);
          }

          String serviceURL = ((Description)stockro.getServiceInstance().
                   getDescription().get(0)).getEndpoint();

          QuoteRequest req = new QuoteRequest(symbol,
               stockro.getResourceID().getResourceID());
          Element elem =  XMLUtils.toDOMDocument(
              req.toString(), debug).getDocumentElement();

          List list = new ArrayList();
          list.add(elem);

          Message msg = new Message(null, secAssertion);
          msg.setSOAPBodies(list);

          Message response = Client.sendRequest(msg, serviceURL, null, null);
          List quotes = response.getBodies();
```

Setting Up Liberty ID-WSF 1.1 Profiles

Federated Access Manager automatically detects which version of the Liberty ID-WSF profiles is being used. If Federated Access Manager is the web services provider (WSP), it detects the version from the incoming SOAP message. If Federated Access Manager is the WSC, it uses the version the WSP has registered with the Discovery Service. If the WSP can not detect the version from the incoming SOAP message or the WSC can not communicate with the Discovery Service, the version defined in the `com.sun.identity.liberty.wsf.version` property in the Federated Access Manager configuration data store will be used. Following are the steps to configure Federated Access Manager to use Liberty ID-WSF 1.1 profiles.

▼ To Configure Federated Access Manager to Use Liberty ID-WSF 1.1 Profiles

1 **Install Federated Access Manager on two different machines.**

Test the installation by logging in to the console at **`http://`***server*`:`*port*`/amserver/UI/Login`.

2 **Setup the two instances of Federated Access Manager for communication using the Liberty ID-FF protocols.**

This entails setting up one instance as the service provider (SP) and the other as the identity provider (IDP). Instructions for doing this can be found in XXXXXEntities and Authentication Domains or in the README file located in the */path-to-context-root*`/samples/liberty/sample1` directory.

Note – This step also entails creating a keystore for each provider. Instructions for this procedure are located in */path-to-context-root*`/samples/saml/xmlsig/keytool.html` or in XXXXXAppendix B, Key Management in this guide. For testing purposes, you can copy the same keystore to each machine; if not, import the public keys from one machine to the other. Be sure to update the Key Alias attribute for each provider in the Federated Access Manager configuration data store and change the cookie name on one of the machines (in the same file) if both machines are in the same domain.

---

3 **Using the Federated Access Manager console on the SP side, change the value of the Protocol Support Enumeration attribute to `urn:liberty:iff:2003-08` in both provider configurations.**

The value of this attribute defines the supported release of the Liberty ID-FF; in this case, version 1.2.

4 Setup the two instances of Federated Access Manager for communication with the Liberty ID-WSF web services.

This entails copying the files located in the */path-to-context-root*/`samples/phase2/wsc` directory to your web container's doc root directory and making the changes specified in the sample README file. The relevant files and corresponding function are:

- `index.jsp`: Retrieves boot strapping resource offering for Discovery Service.
- `discovery-modify.jsp`: Adds resource offering for a user.
- `discovery-query.jsp`: Sends query to Discovery Service for a resource offering.
- `id-sis-pp-modify.jsp`: Sends Data Service Modify request to modify user attributes.
- `id-sis-pp-query.jsp`: Sends Data Service Query request to retrieve user attributes.

5 Copy the `discovery-modify.jsp` reproduced below into the web container's doc root directory.

This JSP is configured to use the Liberty ID-WSF 1.1 Bearer token profile (`<SecurityMechID>urn:liberty:security:2005-02:null:Bearer</SecurityMechID>`) with appropriate directives and should replace the file already in the directory. You can modify this file to use other profiles if you know the defined URI of the particular Liberty ID-WSF 1.1 profile. (X509 or SAML token, for example.)

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

<%@page import="java.io.*,java.util.*,com.sun.identity.saml.common.*,
com.sun.identity.liberty.ws.disco.*,com.sun.identity.liberty.ws.disco.common.*,
javax.xml.transform.stream.*,
com.sun.identity.liberty.ws.idpp.plugin.IDPPResourceIDMapper,
com.iplanet.sso.*,com.sun.liberty.LibertyManager" %>
<html xmlns="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<head><title>Discovery Service Modification</title></head>
<body bgcolor="white">
<h1>Discovery Service Modification</h1>
<%
    if (request.getMethod().equals("GET")) {
        String resourceOfferingFile =
            request.getParameter("discoveryResourceOffering");
        if (resourceOfferingFile == null) {
            resourceOfferingFile= "";
        }
        String entryID =
            request.getParameter("entryID");
        if (entryID == null) {
            entryID= "";
        }
```

```
            // The following three values need to be changed to register a personal
            // profile resource offering for a user.

            String ppProviderID =
             "http://shivalik.red.iplanet.com:58080/amserver/Liberty/idpp";
            String userDN = "uid=amAdmin,ou=People,dc=iplanet,dc=com";
            String ppEndPoint =
           "http://shivalik.red.iplanet.com:58080/amserver/Liberty/idpp";

            String providerID = request.getParameter("providerID");
            String ppResourceID = (new IDPPResourceIDMapper()).getResourceID(
                   ppProviderID, userDN);

            String newPPRO =
                    "<ResourceOffering xmlns=\"urn:liberty:disco:2003-08\">"
                    + "  <ResourceID>" + ppResourceID + "</ResourceID>\n"
                    + "  <ServiceInstance>\n"
                    + "    <ServiceType>urn:liberty:id-sis-pp:2003-08</ServiceType>\n"
                    + "    <ProviderID>" + ppProviderID + "</ProviderID>\n"
                    + "    <Description>"
                    + "      <SecurityMechID>urn:liberty:security:2005-02:null:Bearer"
                    + "</SecurityMechID>\n"
                    + "      <Endpoint>" + ppEndPoint + "</Endpoint>\n"
                    + "    </Description>\n"
                    + "  </ServiceInstance>\n"
                    + "  <Abstract>This is xyz </Abstract>\n"
                    + "</ResourceOffering>";
%>
<form method="POST">
<table>
<tr>
<td>ResourceOffering (for discovery service itself)</td>
<td>
<textarea rows="2" cols="30" name="discoResourceOffering">
<%= resourceOfferingFile %>
</textarea>
</td>
</tr>
<tr>
<td>PP ResourceOffering to add</td>
<td>
<textarea rows="20" cols="60" name="insertStr"><%= newPPRO %></textarea>
</td>
</tr>
<tr>
<td>AND/OR PP ResourceOffering to remove</td>
<td>
```

```
<textarea rows="2" cols="30" name="entryID"></textarea>
</td>
</tr>
</table>
<input type="hidden" name="providerID" value="<%= providerID %>" />
<input type="submit" value="Send Discovery Update Request" />
</form>
<%
    } else {
        try {
            String resourceXMLFile = request.getParameter("discoResourceOffering");
      String resourceXML = null;
            try {
                 BufferedReader bir = new BufferedReader(
          new FileReader(resourceXMLFile));
                 StringBuffer buffer = new StringBuffer(2000);
                 int b1;
                 while ((b1=bir.read ())!= -1) {
                 buffer.append((char) b1);
                 }
                 resourceXML = buffer.toString();
        } catch (Exception e) {
                    %>Warning: cannot read disco resource offering.<%
        }
           String insertString = request.getParameter("insertStr");
           String entryID = request.getParameter("entryID");
           String providerID = request.getParameter("providerID");
           if (resourceXML == null || resourceXML.equals("")) {
               %>ERROR: resource offering missing<%
           } else {
               ResourceOffering offering;
           try {
                offering = new ResourceOffering(DiscoUtils.parseXML(
           resourceXML));
                   DiscoveryClient client = new DiscoveryClient(
            offering,
           SSOTokenManager.getInstance().createSSOToken(request),
           providerID);
                   Modify mod = new Modify();
                   mod.setResourceID(offering.getResourceID());
     mod.setEncryptedResourceID(offering.getEncryptedResourceID());
                   if ((insertString != null) &&
                           !(insertString.equals("")))
                   {
     InsertEntry insert = new InsertEntry(
       new ResourceOffering(
          DiscoUtils.parseXML(insertString)),
              null);
```

```
// Uncommnent the following when it's required.
                        List directives = new ArrayList();
                        Directive dir1 = new Directive(
                          Directive.AUTHENTICATE_REQUESTER);
                        directives.add(dir1);
//                        Directive dir2 = new Directive(
//                          Directive.AUTHORIZE_REQUESTER);
//                        directives.add(dir2);
                        Directive dir3 = new Directive(
                            Directive.GENERATE_BEARER_TOKEN);
                        directives.add(dir3);
                        insert.setAny(directives);
            List inserts = new ArrayList();
            inserts.add(insert);
            mod.setInsertEntry(inserts);
                }
            if ((entryID != null) && !(entryID.equals(""))) {
                        RemoveEntry remove = new RemoveEntry(
                        com.iplanet.am.util.XMLUtils.escapeSpecialCharacters(
                        entryID));
                        List removes = new ArrayList();
                        removes.add(remove);
                        mod.setRemoveEntry(removes);
                    }
                    if ((mod.getInsertEntry() == null) &&
                                (mod.getRemoveEntry() == null))
                    {
                            %>ERROR: empty Modify<%
                    } else {
                        %>
                            <h2>Formed Modify :</h2>
                            <pre><%= SAMLUtils.displayXML(mod.toString()) %></pre>
                        <%
                            ModifyResponse resp2 = client.modify(mod);
                        %>
                            <h2>Got result:</h2>
                            <pre><%= SAMLUtils.displayXML(resp2.toString()) %></pre>
                        <%
                    }
                } catch (Throwable t) {
                    t.printStackTrace();
                    StringWriter buf = new StringWriter();
                    t.printStackTrace(new PrintWriter(buf));
                    %>
                        ERROR: caught exception:
                        <pre>
                    <%
                              out.println(buf.toString());
```

```
                    %>
                        </pre>
                    <%
                }
            }
%>
            <p><a href="index.jsp">Return to index.jsp</a></p>
<%
        } catch (Throwable e) {
            e.printStackTrace();
            StringWriter buf = new StringWriter();
            e.printStackTrace(new PrintWriter(buf));
            %>
                ERROR: oocaught exception:
                <pre>
            <%
                out.println(buf.toString());
            %>
                </pre>
            <%
        }
    }
%>
        <hr/>
    </body>
</html>
```

6 Modify the values of the following properties in the Federated Access Manager configuration data store on the IDP side to reflect the key alias.

- **`com.sun.identity.liberty.ws.wsc.certalias=`***wsc_certificate_alias*
- **`com.sun.identity.liberty.ws.ta.certalias=`***signing_trusted_authority_certificate_alias*
- **`com.sun.identity.liberty.ws.trustedca.certaliases=`***list_of_trusted_authority_certification_*

7 Register the Liberty Personal Profile Service to the user defined by the `userDN` in `discovery-modify.jsp`.

Under the default top-level realm on the instance of Access Manager acting as an IDP, go to Subjects and click User. Select the user and click Services. Click Add and register the Liberty Personal Profile Service.

Note – In the `discovery-modify.jsp` reproduced above, the user is defined as the default administrator, `amAdmin`. See the line:

```
String userDN = "uid=amAdmin,ou=People,dc=iplanet,dc=com";
```

---

8 Restart both instances of Federated Access Manager.

9 **Test that the Liberty ID-WSF 1.1 profiles are working by following the Run the Sample section of the README located in** */path-to-context-root*`/samples/phase2/wsc`**.**

Common Application Programming Interfaces

The following list describes the API common to all Liberty-based Access Manager service components and services.

For more information, including methods and their syntax and parameters, see the *Federated Access Manager 8.0 Java API Reference*.

Common Interfaces

This section summarizes classes that can be used by all Liberty-based Federated Access Manager service components, as well as interfaces common to all Liberty-based Federated Access Manager services. The packages that contain the classes and interfaces are:

`com.sun.identity.liberty.ws.common` Package

This package includes classes common to all Liberty-based Federated Access Manager service components.

TABLE 8–1 `com.sun.identity.liberty.ws.common` Classes

Class	Description
`LogUtil`	Defines methods that are used by the Liberty component of Federated Access Manager to write logs.
`Status`	Represents a common status object.

For more information, including methods and their syntax and parameters, see the *Federated Access Manager 8.0 Java API Reference*.

`com.sun.identity.liberty.ws.interfaces` Package

This package includes interfaces that can be implemented to add their corresponding functionality to each Liberty-based Federated Access Manager web service.

TABLE 8–2 `com.sun.identity.liberty.ws.interfaces` Interfaces

Interface	Description
`Authorizer`	This interface, once implemented, can be used by each Liberty-based web service component for access control. **Note –** The `com.sun.identity.liberty.ws.disco.plugins.DefaultDiscoAuthorizer` class is the implementation of this interface for the Discovery Service. For more information, see XXXXX. The `com.sun.identity.liberty.ws.idpp.plugin.IDPPAuthorizer` class is the implementation for the Liberty Personal Profile Service. For more information, see XXXXX. The `Authorizer` interface enables a web service to check whether a web service consumer (WSC) is allowed to access the requested resource. When a WSC contacts a web service provider (WSP), the WSC conveys a sender identity and an invocation identity. Note that the *invocation identity* is always the subject of the SAML assertion. These conveyances enable the WSP to make an authorization decision based on one or both identities. The Federated Access Manager Policy Service performs the authorization based on defined policies. **Note –** See the *Sun Federated Access Manager 8.0 Technical Overview* for more information about policy management, single sign-on, and user sessions. See the XXXXX for information about creating policy.
`ResourceIDMapper`	This interface is used to map a user DN to the resource identifier associated with it. Federated Access Manager provides implementations of this interface. ■ `com.sun.identity.liberty.ws.disco.plugins.Default64ResourceIDMapper` assumes the Resource ID format to be: *providerID* + "/" + *the Base64 encoded userIDs*. ■ `com.sun.identity.liberty.ws.disco.plugins.DefaultHexResourceIDMapper` assumes the Resource ID format to be: *providerID* + "/" + *the hex string of userID*. ■ `com.sun.identity.liberty.ws.idpp.plugin.IDPPResourceIDMapper` assumes the Resource ID format to be: *providerID* + "/" + *the Base64 encoded userIDs*. A different implementation of the interface may be developed. The implementation class should be given to the provider that hosts the Discovery Service. The mapping between the *providerID* and the implementation class can be configured through the Classes For ResourceIDMapper Plugin attribute.
`ServiceInstanceUpdate`	Interface used to include a SOAP header (`ServiceInstanceUpdateHeader`) when sending a SOAP response.

Common Security API

The Liberty-based security APIs are included in the `com.sun.identity.liberty.ws.security` package and the `com.sun.identity.liberty.ws.common.wsse` package.

`com.sun.identity.liberty.ws.security` Package

Remark 8–1 Reviewer **New**

The `com.sun.identity.liberty.ws.security` package includes the `SecurityTokenProvider` interface for managing Web Service Security (WSS) type tokens and the `SecurityAttributePlugin` interface for inserting security attributes, via an `AttributeStatement`, into the assertion during the Discovery Service token generation. The following table describes the classes used to manage Liberty-based security mechanisms.

TABLE 8–3 `com.sun.identity.liberty.ws.security` Classes

Class	Description
`ProxySubject`	Represents the identity of a proxy, the confirmation key, and confirmation obligation the proxy must possess and demonstrate for authentication purposes.
`ResourceAccessStatement`	Conveys information regarding the accessing entities and the resource for which access is being attempted.
`SecurityAssertion`	Provides an extension to the Assertion class to support ID-WSF `ResourceAccessStatement` and `SessionContextStatement`.
`SecurityTokenManager`	An entry class for the security package `com.sun.identity.liberty.ws.security`. You can call its methods to generate X.509 and SAML tokens for message authentication or authorization. It is designed as a provider model, so different implementations can be plugged in if the default implementation does not meet your requirements.
`SecurityUtils`	Defines methods that are used to get certificates and sign messages.
`SessionContext`	Represents the session status of an entity to another system entity.
`SessionContextStatement`	Conveys the session status of an entity to another system entity within the body of an `<saml:assertion>` element.
`SessionSubject`	Represents a Liberty subject with its associated session status.

For more information, including methods and their syntax and parameters, see the *Federated Access Manager 8.0 Java API Reference*.

`com.sun.identity.liberty.ws.common.wsse` Package

This package includes classes for creating security tokens used for authentication and authorization in accordance with the *Liberty ID-WSF Security Mechanisms*. Both WSS X.509 and SAML tokens are supported.

TABLE 8–4 `com.sun.identity.liberty.ws.common.wsse` Classes

Class	Description
`BinarySecurityToken`	Provides an interface to parse and create the X.509 Security Token depicted by Web Service Security: X.509
`WSSEConstants`	Defines constants used in security packages.

For more information, including methods and their syntax and parameters, see the *Federated Access Manager 8.0 Java API Reference*.

Web Service Consumer Sample

The `wsc` directory contains a collection of files to deploy and run a web service consumer (WSC).

Note – Before implementing this sample, you must have two instances of Federated Access Manager installed, and running, and Liberty-enabled. Completing the procedure in XXXXXsample1 Directory will accomplish this.

In addition, this sample illustrates how to use the Discovery Service and Data Services Template client APIs to allow the WSC to communicate with a web service provider (WSP). This sample describes the flow of the Liberty-based Web Service Framework (ID-WSF) and how the security mechanisms and interaction service are integrated. The `Readme.html` file in the `wsc` directory provides detailed steps on how to deploy and configure this sample. For more information, see also XXXXXChapter 7, Data Services and XXXXXChapter 8, Discovery Service.

Authentication Web Service

The SOAP specifications define an XML-based messaging paradigm, but do not specify any particular security mechanisms. Particularly, they do not describe user authentication using SOAP messages. To rectify this, the Authentication Web Service was implemented based on the *Liberty ID-WSF Authentication Service and Single Sign-On Service Specification*. The specification defines a protocol that adds the Simple Authentication and Security Layer (SASL) authentication functionality to the SOAP binding described in the *Liberty ID-WSF SOAP Binding Specification* and, XXXXXXChapter 9, SOAP Binding Service in this guide. The Authentication Web Service is for provider-to-provider authentication.

Note – The specification also contains an XML schema that defines the authentication protocol. More information can be found in XXXXXXSchema Files and Service Definition Documents.

Authentication Web Service Default Implementation

The Authentication Web Service attribute is a *global* attribute. The value of this attribute is carried across the Access Manager configuration and inherited by every organization. The attribute for the Authentication Web Service is defined in the XML service file `amAuthnSvc.xml` service file and is called the Mechanism Handlers List.

Note – For information about the types of attributes used in Access Manager, see the *Sun Java System Access Manager 7.1 Technical Overview*. For information about service files, see the *Sun Java System Access Manager 7.1 Administration Guide*.

Mechanism Handlers List

The Mechanism Handler List attribute stores information about the SASL mechanisms that are supported by the Authentication Web Service.

`key` Parameter

The required `key` defines the SASL mechanism supported by the Authentication Web Service.

`class` Parameter

The required `class` specifies the name of the implemented class for the SASL mechanism. Two authentication mechanisms are supported by the following default implementations:

TABLE 8–5 Default Implementations for Authentication Mechanism

Class	Description
`com.sun.identity.liberty.ws.authnsvc.mechanism.PlainMechanismHandler`	This class is the default implementation for the PLAIN authentication mechanism. It maps user identifiers and passwords in the PLAIN mechanism to the user identifiers and passwords in the LDAP authentication module under the root organization.
`com.sun.identity.liberty.ws.authnsvc.mechanism.CramMD5MechanismHandler`	This class is the default implementation for the CRAM-MD5 authentication mechanism.

Note – The Authentication Web Service layer provides an interface that must be implemented for each SASL mechanism to process the requested message and return a response. For more information, see XXXXXcom.sun.identity.liberty.ws.authnsvc.mechanism Package.

Authentication Web Service API

The Authentication Web Service provides programmatic interfaces to allow clients to interact with it. The following sections provide short descriptions of these packages. For more detailed information, see the Java API Reference in */AccessManager-base*`/SUNWam/docs` or on docs.sun.com. The authentication-related packages include:

- "`com.sun.identity.liberty.ws.authnsvc` Package" on page 171
- "`com.sun.identity.liberty.ws.authnsvc.mechanism` Package" on page 171
- "`com.sun.identity.liberty.ws.authnsvc.protocol` Package" on page 172

`com.sun.identity.liberty.ws.authnsvc` Package

This package provides web service clients with a method to request authentication credentials from the Authentication Web Service and receive responses back from it using the Simple Authentication and Security Layer (SASL).

`com.sun.identity.liberty.ws.authnsvc.mechanism` Package

This package provides an interface that must be implemented for each different SASL mechanism to enable authentication using them. Each SASL mechanism will correspond to one implementation that will process incoming SASL requests and generate outgoing SASL responses.

`com.sun.identity.liberty.ws.authnsvc.protocol` Package

This package provides classes that correspond to the request and response elements defined in the Liberty XSD schema that accompanies the *Liberty ID-WSF Authentication Service Specification*. More information about the XSD schemas can be found in XXXXXSchema Files and Service Definition Documents.

Access the Authentication Web Service

The URL to gain access to the Authentication Web Service is:

`http://`*SERVER_HOST*`:`*SERVER_PORT*`/`*SERVER_DEPLOY_URI*`/Liberty/authnsvc`

This URL is normally used by the Access Manager client API to access the service. For example, the Access Manager public client, `com.sun.identity.liberty.ws.authnsvc.AuthnSvcClient` uses this URL to authenticate principals with Access Manager.

Authentication Web Service Sample

A sample authentication client is included with Access Manager. It is located in the */AccessManager-base*`/SUNWam/samples/phase2/authnsvc` directory. The client uses the `PLAIN` SASL authentication mechanism. It first authenticates against the Authentication Web Service, then extracts a resource offering to bootstrap the Discovery Service. It looks for a SAML `Bearer` token credential, issues a discovery query request with SAML assertion included, and receives a response. The `Readme.html` file in the sample directory provides detailed steps on how to deploy and configure this sample.

Note – This sample can be used by a Liberty User Agent Device WSC.

Data Services

A *data service* is a web service that supports the query and modification of data regarding a principal. An example of a data service is a web service that hosts and exposes a principal's profile information, such as name, address and phone number. A *query* is when a web service consumer (WSC) requests and receives the data (in XML format). A *modify* is when a WSC sends new information to update the data. The Liberty Alliance Project has defined the *Liberty ID-WSF Data Services Template Specification* (Liberty ID-WSF-DST) as the standard protocol for the query and modification of data profiles exposed by a data service. Using this specification, the Liberty Alliance Project has developed additional specifications for other types of data services: personal profile service, geolocation service, contact service, and calendar

service). Of these data services, Access Manager has implemented the Liberty Personal Profile Service and, using the included sample, the Liberty Employee Profile Service.

Liberty Personal Profile Service

The Liberty Personal Profile Service is a default Access Manager identity service. It can be queried for identity data and its attributes can be updated.

For access to occur, the hosting provider of the Liberty Personal Profile Service needs to be registered with the Discovery Service on behalf of each identity principal. To register a service with the Discovery Service, update a resource offering for that service. For more information, see XXXXXChapter 8, Discovery Service.

The URL to gain access to the Liberty Personal Profile Service is:

`http://`*SERVER_HOST*`:`*SERVER_PORT*`/`*SERVER_DEPLOY_URI*`/Liberty/idpp`

This URL is normally used by the Access Manager client API to access the service. For example, the Access Manager public Data Service Template client, `com.sun.identity.liberty.ws.dst.DSTClient` uses this URL to query and modify an identity's personal profile attributes stored in Access Manager.

Liberty Employee Profile Service

The Liberty Employee Profile Service sample provides a collection of files that can be used to deploy and invoke a new corporate data service. The files are located in the */path-to-context-root*`/fam/samples/phase2/sis-ep` directory.

Note – Before implementing this sample, you must have two instances of Federated Access Manager installed, and running, and Liberty-enabled. Completing the procedure in XXXXXsample1 Directory will accomplish this.

The Liberty Employee Profile Service is a deployment of the *Liberty ID-SIS Employee Profile Service Specification* (ID-SIS-EP), which is one of the *Liberty Alliance ID-SIS 1.0 Specifications*. The `Readme.html` file in the sample directory provides detailed steps on how to deploy and configure this sample. For more information, see also XXXXXChapter 7, Data Services

Data Services Template API

Access Manager contains two packages based on the Liberty ID-WSF-DST. They are:

- "`com.sun.identity.liberty.ws.dst` Package" on page 174
- "`com.sun.identity.liberty.ws.dst.service` Package" on page 174

For more detailed API documentation, including methods and their syntax and parameters, see the Java API Reference in */AccessManager-base*`/SUNWam/docs` or on docs.sun.com.

`com.sun.identity.liberty.ws.dst` Package

The following table summarizes the classes in the Data Services Template client API that are included in the `com.sun.identity.liberty.ws.dst` package.

TABLE 8–6 Data Service Client APIs

Class	Description
`DSTClient`	Provides common functions for the Data Services Templates query and modify options.
`DSTData`	Provides a wrapper for any data entry.
`DSTModification`	Represents a Data Services Template modification operation.
`DSTModify`	Represents a Data Services Template modify request.
`DSTModifyResponse`	Represents a Data Services Template response to a DST modify request.
`DSTQuery`	Represents a Data Services Template query request.
`DSTQueryItem`	Wrapper for one query item.
`DSTQueryResponse`	Represents a Data Services Template query response.
`DSTUtils`	Provides utility methods used by the DST layer.

`com.sun.identity.liberty.ws.dst.service` Package

This package provides a handler class that can be used by any generic identity data service that is built using the *Liberty Alliance ID-SIS Specifications*.

Note – The Liberty Personal Profile Service is built using the *Liberty ID-SIS Personal Profile Service Specification*, based on the *Liberty Alliance ID-SIS Specifications*.

The `DSTRequestHandler` class is used to process query or modify requests sent to an identity data service. It is an implementation of the interface `com.sun.identity.liberty.ws.soapbinding.RequestHandler`. For more detailed API documentation, see the Java API Reference in */AccessManager-base*`/SUNWam/docs` or on docs.sun.com.

Note – Access Manager provides a sample that makes use of the `DSTRequestHandler` class. The `sis-ep` sample illustrates how to implement the `DSTRequestHandler` and deploy a new identity data service instance. The sample is located in the */AccessManager-base*`/SUNWam/samples/phase2/sis-ep` directory.

Discovery Service

Sun Java™ System Access Manager contains a Discovery Service defined by the Liberty Alliance Project. The Discovery Service allows a requesting entity to dynamically determine a principal's registered identity service. It might also function as a security token service, issuing security tokens to the requester that can then be used in the request to the discovered identity service.

Generating Security Tokens

Remark 8–2 Reviewer **New section.**

In general, a discovery service and an identity provider are hosted on the same machine. Because the identity provider hosting the Discovery Service might be fulfilling other roles for an identity (such as a Policy Decision Point or an Authentication Authority), it can be configured to provide the requesting entity with security tokens. The Discovery Service can include a security token (inserted into a SOAP message header) in a `DiscoveryLookup` response. The token can then be used as a credential to invoke the service returned with it.

▼ To Configure the Discovery Service to Generate Security Tokens

After completing the following procedure, you can test the functionality by running the samples.

1 Generate the keystore and certificate aliases for the machines that are hosting the Discovery Service, the WSP and the WSC.

Access Manager uses a Java keystore for storing the public and private keys so, if this is a new deployment, you might need to generate one. `keystore.html` in `AccessManager-base/SUNWam/samples/saml/xmlsig/` has information on accomplishing this using `keytool`, the key and certificate management utility supplied with the Java Platform, Standard Edition. In short, `keytool` generates key pairs as separate key entries (one for a public

key and the other for its associated private key). It wraps the public key into an X.509 self-signed certificate (one for which the issuer/signer is the same as the subject), and stores it as a single-element certificate chain. Additionally, the private key is stored separately, protected by a password, and associated with the certificate chain for the corresponding public key. All public and private keystore entries are accessed via unique aliases.

2 Update the values of the following key-related properties in the `AMConfig.properties` files of the appropriate deployed instances of Access Manager.

`AMConfig.properties` is located in `/etc/opt/SUNWam/config/`.

Note – The same property might have already been edited depending on the deployment scenario.

---

a. Update the values of the following key-related properties in the `AMConfig.properties` files on the machine that hosts the Discovery Service.

- `com.sun.identity.saml.xmlsig.keystore` defines the location of the keystore file.
- `com.sun.identity.saml.xmlsig.storepass` defines the location of the file that contains the password used to access the keystore file.
- `com.sun.identity.saml.xmlsig.keypass` defines the location of the file that contains the password used to protect the private key of a generated key pair.
- `com.sun.identity.liberty.ws.ta.certalias` defines the certificate alias used by the Discovery Service to sign SAML assertions.
- `com.sun.identity.liberty.ws.wsc.certalias` defines the certificate alias used by the Discovery Service to sign the protocol response.

b. Update the values of the following key-related properties in the `AMConfig.properties` files on the machines that acts as the WSP.

- `com.sun.identity.saml.xmlsig.keystore` defines the location of the keystore file.
- `com.sun.identity.saml.xmlsig.storepass` defines the location of the file that contains the password used to access the keystore file.
- `com.sun.identity.saml.xmlsig.keypass` defines the location of the file that contains the password used to protect the private key of a generated key pair.
- `com.sun.identity.liberty.ws.wsc.certalias` defines the certificate alias used for signing the WSP protocol responses.
- `com.sun.identity.liberty.ws.trustedca.certaliases` defines the certificate alias and the Provider ID list on which the WSP is trusting.

c. Update the values of the following key-related properties in the `AMConfig.properties` files on the machine that acts as the WSC.

- `com.sun.identity.saml.xmlsig.keystore` defines the location of the keystore file.

- `com.sun.identity.saml.xmlsig.storepass` defines the location of the file that contains the password used to access the keystore file.
- `com.sun.identity.saml.xmlsig.keypass` defines the location of the file that contains the password used to protect the private key of a generated key pair.
- `com.sun.identity.liberty.ws.wsc.certalias` defines the certificate alias used by web service clients for signing protocol requests.

Note – The `com.sun.identity.liberty.ws.wsc.certalias` property must be *added* to the `AMConfig.properties` file.

---

3 Configure each identity provider and service provider as an entity using the Access Manager Federation module.

This entails configuring a entity for each provider using the Access Manager Console or loading an XML metadata file using `amadmin`. See XXXXXEntities for information on the former and Chapter 1, "The amadmin Command Line Tool," in *Sun Java System Access Manager 7.1 Administration Reference* for information on the latter.

Note – Be sure to configure each provider's entity so that all providers in the scenario are defined as Trusted Providers.

---

4 Establish provider trust between the entities by creating an authentication domain using the Access Manager Federation module.

See XXXXXAuthentication Domains.

5 Change the default value of the Provider ID for the Discovery Service on the machine where the Discovery Service is hosted to the value that reflects the previously loaded metadata.

a. Click the Web Services tab from the Access Manager Console.

b. Click the Discovery Service tab under Web Services.

c. Change the default value of the Provider ID from *protocol*`://`*host*`:`*port*/*deployuri*`/Liberty/disco`.

Note – If using the samples, make sure that the value of Provider ID in `discovery-modify.jsp` is changed, if necessary, before the WSP registers with the Discovery Service.

---

6 **Change the default value of the Provider ID for the Liberty Personal Profile Service on the machine where the Liberty Personal Profile Service is hosted to the value that reflects the previously loaded metadata.**

   a. **Click the Web Services tab from the Access Manager Console.**

   b. **Click the Liberty Personal Profile Service tab under Web Services.**

   c. **Change the default value of the Provider ID from** *protocol*`://`*host*`:`*port*/*deployuri*`/Liberty/idpp`**.**

7 **Register a resource offering for the WSP using either of the following methods.**

- Access Manager Console

  See XXXXXXStoring Resource Offerings for information on registering a resource offering using the Access Manager Console.

- Client API

  See `discovery-modify.jsp` in *AccessManager-base*`/samples/phase2/wsc` which calls the API for registering a resource offering.

Also, make sure that the appropriate directives are chosen.

- For SAML Bearer token use `GenerateBearerToken` or `AuthenticateRequester`.
- For SAML Token (Holder of key) use `AuthenticateRequester` or `AuthorizeRequester`.

Discovery Service APIs

Access Manager contains several Java packages that are used by the Discovery Service. They include:

- `com.sun.identity.liberty.ws.disco` includes a client API that provides interfaces to communicate with the Discovery Service. See "Client APIs in `com.sun.identity.liberty.ws.disco`" on page 179.
- `com.sun.identity.liberty.ws.disco.plugins` includes an interface that can be used to develop plug-ins. The package also contains some default plug-ins. See "`com.sun.identity.liberty.ws.disco.plugins.DiscoEntryHandler` Interface" on page 180.
- `com.sun.identity.liberty.ws.interfaces` includes interfaces that can be used to implement functionality common to all Liberty-enabled identity services. Several implementations of these interfaces have been developed for the Discovery Service. See "`com.sun.identity.liberty.ws.interfaces.Authorizer` Interface" on page 180 and "`com.sun.identity.liberty.ws.interfaces.ResourceIDMapper` Interface" on page 182.

Note – Additional information is in the Java API Reference in */AccessManager-base*/SUNWam/docs or on docs.sun.com. Information about the com.sun.identity.liberty.ws.common package is in XXXXXXCommon Service Interfaces in XXXXXChapter 11, Application Programming Interfaces.

Client APIs in com.sun.identity.liberty.ws.disco

The following table summarizes the client APIs in the package com.sun.identity.liberty.ws.disco. For more information, including methods and their syntax and parameters, see the Java API Reference in */AccessManager-base*/SUNWam/docs or on docs.sun.com.

TABLE 8–7 Discovery Service Client APIs

Class	Description
Description	Represents a DescriptionType element of a service instance.
Directive	Represents a discovery service DirectiveType element.
DiscoveryClient	Provides methods to send Discovery Service queries and modifications.
EncryptedResourceID	Represents an EncryptionResourceID element for the Discovery Service.
InsertEntry	Represents an Insert Entry for Discovery Modify request.
Modify	Represents a discovery modify request.
ModifyResponse	Represents a discovery response to a modify request.
Query	Represents a discovery Query object.
QueryResponse	Represents a response to a discovery query request.
RemoveEntry	Represents a remove entry element for the discovery modify request.
RequestedService	Enables the requester to specify that all the resource offerings returned must be offered through a service instance that complys with one of the specified service types.
ResourceID	Represents a Discovery Service Resource ID.
ResourceOffering	Associates a resource with a service instance that provides access to that resource.
ServiceInstance	Describes a web service at a distinct protocol endpoint.

`com.sun.identity.liberty.ws.disco.plugins.DiscoEntryHandler` Interface

This interface is used to get and set discovery entries for a user. A number of default implementations are provided, but if you want to handle this function differently, implement this interface and set the implementing class as the value of the Entry Handler Plugin Class attribute as discussed in XXXXXEntry Handler Plug-in Class. The default implementations of this interface are described in the following table.

TABLE 8–8 Implementations of `com.sun.identity.liberty.ws.disco.plugins.DiscoEntryHandler`

Class	Description
`UserDiscoEntryHandler`	Gets or modifies discovery entries stored in the user's entry as a value of the `sunIdentityServerDiscoEntries` attribute. The `UserDiscoEntryHandler` implementation is used in business-to-consumer scenarios such as the Liberty Personal Profile Service.
`DynamicDiscoEntryHandler`	Gets discovery entries stored as a value of the `sunIdentityServerDynamicDiscoEntries` dynamic attribute in the Discovery Service. Modification of these entries is not supported and always returns `false`. The resource offering is saved in an organization or a role. The `DynamicDiscoEntryHandler` implementation is used in business-to-business scenarios such as the Liberty Employee Profile service.
`UserDynamicDiscoEntryHandler`	Gets a union of the discovery entries stored in the user entry `sunIdentityServerDiscoEntries` attribute and discovery entries stored in the Discovery Service `sunIdentityServerDynamicDiscoEntries` attribute. It modifies only discovery entries stored in the user entry. The `UserDynamicDiscoEntryHandler` implementation can be used in both business-to-consumer and business-to-business scenarios.

`com.sun.identity.liberty.ws.interfaces.Authorizer` Interface

This interface is used to enable an identity service to check the authorization of a WSC. The `DefaultDiscoAuthorizer` class is the default implementation of this interface. The class uses the Access Manager Policy Service for creating and applying policy definitions. Policy definitions for the Discovery Service are configured using the Access Manager Console.

Note – The Policy Service looks for an `SSOToken` defined for Authenticated Users or Web Service Clients. For more information on this and the Policy Service in general, see the *Sun Java System Access Manager 7.1 Administration Guide*.

▼ To Configure Discovery Service Policy Definitions

1. **In the Access Manager Console, click the Access Control tab.**
2. **Select the name of the realm in which the policy definitions will be configured.**
3. **Select Policies to access policy configurations.**
4. **Click New Policy to add a new policy definition.**
5. **Type a name for the policy.**
6. **(Optional) Enter a description for the policy.**
7. **(Optional) Select the check box next to Active.**
8. **Click New to add rules to the policy definition.**
9. **Select Discovery Service for the rule type and click Next.**
10. **Type a name for the rule.**
11. **Type a resource on which the rule acts.**

    The Resource Name field uses the form *ServiceType* + *RESOURCE_SEPARATOR* + *ProviderID*. For example, `urn:liberty:id-sis-pp:2003-08;http://example.com`.
12. **Select an action and appropriate value for the rule.**

    Discovery Service policies can only look up or update data.
13. **Click Finish to configure the rule.**

    The `com.sun.identity.liberty.ws.interfaces.Authorizer` interface can be implemented by any web service in Access Manager. For more information, see XXXXXCommon Service Interfaces and the Java API Reference in */AccessManager-base*`/SUNWam/docs` or on docs.sun.com.
14. **Click New to add subjects to the policy definition.**
15. **Select the subject type and click Next.**

16. **Type a name for the group of subjects.**
17. **(Optional) Click the check box if this is an exclusive group.**
18. **Select the users and click to add them to the group.**
19. **Click Finish to return to the policy definition screen.**
20. **Click New to add conditions to the policy definition.**
21. **Select the condition type and click Next.**
22. **Type values for the displayed attributes.**
    For more information, see the *Sun Java System Access Manager 7.1 Administration Guide*.
23. **Click Finish to return to the policy definition screen.**
24. **Click New to add response providers to the policy definition.**
25. **Type a name for the response provider.**
26. **(Optional) Add values for the StaticAttribute.**
27. **(Optional) Add values for the DynamicAttribute.**
28. **Click Finish to return to the policy definition screen.**
29. **Click Create to finish the policy configuration.**

`com.sun.identity.liberty.ws.interfaces.ResourceIDMapper` Interface

This interface is used to map a user ID to the resource identifier associated with it. Access Manager provides two implementations of this interface.

- `com.sun.identity.liberty.ws.disco.plugins.Default64ResourceIDMapper` assumes the format to be *providerID* + "/" + *the Base64 encoded userIDs*
- `com.sun.identity.liberty.ws.disco.plugins.DefaultHexResourceIDMapper` assumes the format to be *providerID* + "/" + *the hex string of userIDs*

A different implementation of the interface may be developed. The implementation class should be given to the provider that hosts the Discovery Service. The mapping between the *providerID* and the implementation class can be configured through the XXXXXClasses For ResourceIDMapper Plug-in attribute.

Note – The `com.sun.identity.liberty.ws.interfaces.ResourceIDMapper` interface is common to all identity services in Access Manager not only the Discovery Service. For more information, see XXXXXCommon Service Interfaces and the Java API Reference in */AccessManager-base*`/SUNWam/docs` or on docs.sun.com.

Access the Discovery Service

The URL to gain access to the Discovery Service is:

`http://`*SERVER_HOST*`:`*SERVER_PORT*`/`*SERVER_DEPLOY_URI*`/Liberty/disco`

This URL is normally used by the Access Manager client API to access the service. For example, the Access Manager public Discovery Service client, `com.sun.identity.liberty.ws.disco.DiscoveryClient` uses this URL to query and modify the resource offerings of an identity.

Discovery Service Sample

A sample that shows the process for querying and modifying the Discovery Service is included with Access Manager in the */AccessManager-base*`/SUNWam/samples/phase2/wsc` directory. The sample initially shows how to deploy and run a WSC. The final portion queries the Discovery Service and modifies identity data in the Liberty Personal Profile Service.

SOAP Binding Service

Sun™ Java System Access Manager contains an implementation of the *Liberty ID-WSF SOAP Binding Specification* from the Liberty Alliance Project. The specification defines a transport layer for sending and receiving SOAP messages.

- "`SOAPReceiver` Servlet" on page 183
- "SOAP Binding Service Package" on page 184

`SOAPReceiver` Servlet

The `SOAPReceiver` servlet receives a `Message` object from a web service client (WSC), verifies the signature, and constructs its own `Message` object for processing by Access Manager. The `SOAPReceiver` then invokes the correct request handler class to pass this second `Message` object on to the appropriate Access Manager service for a response. When the response is generated, the `SOAPReceiver` returns this `Message` object back to the WSC. More information can be found in the XXXXXSOAP Binding Process.

SOAP Binding Service Package

The Access Manager SOAP Binding Service includes a Java package named `com.sun.identity.liberty.ws.soapbinding`. This package provides classes to construct SOAP requests and responses and to change the contact point for the SOAP binding. The following table describes some of the available classes. For more detailed information, see the Java API Reference in */AccessManager-base*`/SUNWam/docs` or on docs.sun.com.

TABLE 8–9 SOAP Binding Service API

Class	Description
`Client`	Provides a method with which a WSC can send a request to a WSP using a SOAP connection. It also returns the response.
`ConsentHeader`	Represents the SOAP element named `Consent`.
`CorrelationHeader`	Represents the SOAP element named `Correlation`. By default, `CorrelationHeader` will always be signed.
`ProcessingContextHeader`	Represents the SOAP element named `ProcessingContext`.
`ProviderHeader`	Represents the SOAP element named `Provider`.
`RequestHandler`	Defines an interface that needs to be implemented on the server side by each web service in order to receive a request from a WSC and generate a response. After implementing the class, it must be registered in the SOAP Binding Service so the SOAP framework knows where to forward incoming requests.
`Message`	Represents a SOAP message and is used by both the web service client and server to construct SOAP requests and responses. Each SOAP message has multiple headers and bodies. It may contain a certificate for client authentication, the IP address of a remote endpoint, and a SAML assertion used for signing.
`ServiceInstanceUpdateHeader`	Allows a service to change the endpoint on which requesters will contact it.
`ServiceInstanceUpdateHeader.Credential`	Allows a service to use a different security mechanism and credentials to access the requested resource.
`SOAPFault`	Represents the SOAP element named `SOAP Fault`.
`SOAPFaultDetail`	Represents the SOAP element named `Detail`, a child element of `SOAP Fault`.
`UsageDirectiveHeader`	Defines the SOAP element named `UsageDirective`.

See "PAOS Binding" on page 187 for information on this reverse HTTP binding for SOAP.

Interaction Service

Providers of identity services often need to interact with the owner of a resource to get additional information, or to get their consent to expose data. The Liberty Alliance Project has defined the *Liberty ID-WSF Interaction Service Specification* to specify how these interactions can be carried out. Of the options defined in the specification, Federated Access Manager has implemented the Interaction RequestRedirect Profile. In this profile, the WSP requests the connecting WSC to redirect the user agent (principal) to an interaction resource (URL) at the WSP. When the user agent sends an HTTP request to get the URL, the WSP has the opportunity to present one or more pages to the principal with questions for other information. After the WSP obtains the information it needs to serve the WSC, it redirects the user agent back to the WSC, which can now reissue its original request to the WSP.

Configuring the Interaction Service

While there is no XML service file for the Interaction Service, this service does have properties. The properties are configured upon installation in the `AMConfig.properties` file located in */path-to-context-root*`/fam/lib` and are described in the following table.

TABLE 8–10 Interaction Service Properties in `AMConfig.properties`

Property	Description
`com.sun.identity.liberty.interaction.wspRedirectHandler`	Points to the URL where the `WSPRedirectHandler` servlet is deployed. The servlet handles the service provider side of interactions for user redirects.
`com.sun.identity.liberty.interaction.wscSpecifiedInteractionChoice`	Indicates the level of interaction in which the WSC will participate if the WSC participates in user redirects. Possible values include `interactIfNeeded`, `doNotInteract`, and `doNotInteractForData`. The affirmative `interactIfNeeded` is the default.
`com.sun.identity.liberty.interaction.wscWillIncludeUserInteractionHeader`	Indicates whether the WSC will include a SOAP header to indicate certain preferences for interaction based on the Liberty specifications. The default value is `yes`.
`com.sun.identity.liberty.interaction.wscWillRedirect`	Indicates whether the WSC will participate in user redirections. The default value is `yes`.

TABLE 8–10 Interaction Service Properties in `AMConfig.properties` *(Continued)*

Property	Description
`com.sun.identity.liberty.interaction.wscSpecifiedMaxInteractionTime`	Indicates the maximum length of time (in seconds) the WSC is willing to wait for the WSP to complete its portion of the interaction. The WSP will not initiate an interaction if the interaction is likely to take more time than . For example, the WSP receives a request where this property is set to a maximum 30 seconds. If the WSP property `com.sun.identity.liberty.interaction.wspRedirectTime` is set to 40 seconds, the WSP returns a SOAP fault (`timeNotSufficient`), indicating that the time is insufficient for interaction.
`com.sun.identity.liberty.interaction.wscWillEnforceHttpsCheck`	Indicates whether the WSC will enforce HTTPS in redirected URLs. The Liberty Alliance Project specifications state that, the value of this property is always `yes`, which indicates that the WSP will not redirect the user when the value of `redirectURL` (specified by the WSP) is not an HTTPS URL. The `false` value is primarily meant for ease of deployment in a phased manner.
`com.sun.identity.liberty.interaction.wspWillRedirect`	Initiates an interaction to get user consent for something or to collect additional data. This property indicates whether the WSP will redirect the user for consent. The default value is `yes`.
`com.sun.identity.liberty.interaction.wspWillRedirectForData`	Initiates an interaction to get user consent for something or to collect additional data. This property indicates whether the WSP will redirect the user to collect additional data. The default value is `yes`.
`com.sun.identity.liberty.interaction.wspRedirectTime`	Indicates the length of time (in seconds) that the WSP expects to take to complete an interaction and return control back to the WSC. For example, the WSP receives a request indicating that the WSC will wait a maximum 30 seconds (set in `com.sun.identity.liberty.interaction.wscSpecifiedMaxInteractionTime`) for interaction. If the `wspRedirectTime` is set to 40 seconds, the WSP returns a SOAP fault (`timeNotSufficient`), indicating that the time is insufficient for interaction.
`com.sun.identity.liberty.interaction.wspWillEnforceHttpsCheck`	Indicates whether the WSP will enforce a HTTPS `returnToURL` specified by the WSC. The Liberty Alliance Project specifications state that the value of this property is always `yes`. The `false` value is primarily meant for ease of deployment in a phased manner.

TABLE 8–10 Interaction Service Properties in `AMConfig.properties` *(Continued)*

Property	Description
`com.sun.identity.liberty.interaction.wspWillEnforceReturnToHostEqualsRequestHost`	Indicates whether the WSP would enforce the address values of `returnToHost` and `requestHost` if they are the same. The Liberty Alliance Project specifications state that the value of this property is always `yes`. The `false` value is primarily meant for ease of deployment in a phased manner.
`com.sun.identity.liberty.interaction.htmlStyleSheetLocation`	Points to the location of the style sheet that is used to render the interaction page in HTML.
`com.sun.identity.liberty.interaction.wmlStyleSheetLocation`	Points to the location of the style sheet that is used to render the interaction page in WML.

Interaction Service API

The Federated Access Manager Interaction Service includes a Java package named `com.sun.identity.liberty.ws.interaction`. WSCs and WSPs use the classes in this package to interact with a resource owner. The following table describes the classes.

TABLE 8–11 Interaction Service Classes

Class	Description
`InteractionManager`	Provides the interface and implementation for resource owner interaction.
`InteractionUtils`	Provides some utility methods related to resource owner interaction.
`JAXBObjectFactory`	Contains factory methods that enable you to construct new instances of the Java representation for XML content.

For more information, including methods and their syntax and parameters, see the *Federated Access Manager 8.0 Java API Reference*.

PAOS Binding

Federated Access Manager has implemented the optional *Liberty Reverse HTTP Binding for SOAP Specification*. This specification defines a message exchange protocol that permits an HTTP client to be a SOAP responder. HTTP clients are no longer necessarily equipped with HTTP servers. For example, mobile terminals and personal computers contain web browsers yet they do not operate HTTP servers. These clients, though, can use their browsers to interact

with an identity service, possibly a personal profile service or a calendar service. These identity services could also be beneficial when the client devices interact with an HTTP server. The use of PAOS makes it possible to exchange information between user agent-hosted services and remote servers. This is why the reverse HTTP for SOAP binding is also known as PAOS; the spelling of SOAP is reversed.

Comparison of PAOS and SOAP

In a typical SOAP binding, an HTTP client interacts with an identity service through a client request and a server response. For example, a cell phone user (client) can contact the phone service provider (service) to retrieve stock quotes and weather information. The service verifies the user's identity and responds with the requested information.

In a reverse HTTP for SOAP binding, the phone service provider plays the client role, and the cell phone client plays the server role. The initial SOAP request from the server is actually bound to an HTTP response. The subsequent response from the client is bound to a request.

PAOS Binding API

The Federated Access Manager implementation of PAOS binding includes a Java package named `com.sun.identity.liberty.ws.paos`. This package provides classes to parse a PAOS header, make a PAOS request, and receive a PAOS response.

Note – This API is used by PAOS clients on the HTTP server side. An API for PAOS servers on the HTTP client side would be developed by the manufacturers of the HTTP client side products, for example, cell phone manufacturers.

The following table describes the available classes in `com.sun.identity.liberty.ws.paos`. For more detailed API documentation, see the *Federated Access Manager 8.0 Java API Reference*.

TABLE 8–12 PAOS Binding Classes

Class	Description
`PAOSHeader`	Used by a web application on the HTTP server side to parse a PAOS header in an HTTP request from the user agent side.

TABLE 8–12 PAOS Binding Classes *(Continued)*

Class	Description
`PAOSRequest`	Used by a web application on the HTTP server side to construct a PAOS request message and send it via an HTTP response to the user agent side. **Note –** `PAOSRequest` is made available in `PAOSResponse` to provide correlation, if needed, by API users.
`PAOSResponse`	Used by a web application on the HTTP server side to receive and parse a PAOS response using an HTTP request from the user agent side.
`PAOSException`	Represents an error occurring while processing a SOAP request and response.

PAOS Binding Sample

A sample that demonstrates PAOS service interaction between an HTTP client and server is provided in the */path-to-context-root/*`fam/samples/phase2/paos` directory. The `paos` directory contains a collection of files that demonstrate how to set up and invoke a PAOS Service interaction between a client and server. The sample is based on the following scenario: a cell phone user subscribes to a news service offered by the cell phone's manufacturer. The news service automatically provides stocks and weather information to the user's cell phone at regular intervals. In this scenario, the manufacturer is the news service provider, and the individual cell phone user is the consumer. The PAOS client is a servlet, and the PAOS server is a stand-alone Java program. (In an actual deployment, the server-side code would be developed by a service provider.) Instructions on how to run the sample can be found in the `Readme.html` or `Readme.txt` file. Both files are included in the `paos` directory. After running the sample, you will see the output from the PAOSServer program.

Note – You can also see the output from PAOSClientServlet program in the log file of the Web Server. For example, when using Sun Java System Web Server, look in the `log` subdirectory for the errors file.

The following code example is the PAOS client servlet.

EXAMPLE 8–1 PAOS Client Servlet From PAOS Sample

```
import java.util.*;
import java.io.*;

import javax.servlet.*;
import javax.servlet.http.*;
```

EXAMPLE 8–1 PAOS Client Servlet From PAOS Sample *(Continued)*

```
import com.sun.identity.liberty.ws.paos.*;

import com.sun.identity.liberty.ws.idpp.jaxb.*;

public class PAOSClientServlet extends HttpServlet {

  public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

      PAOSHeader paosHeader = null;
      try {
    paosHeader = new PAOSHeader(req);
      } catch (PAOSException pe1) {
    pe1.printStackTrace();

    String msg = "No PAOS header\\n";
    res.setContentType("text/plain");
    res.setContentLength(1+msg.length());
    PrintWriter out = new PrintWriter(res.getOutputStream());
    out.println(msg);
    out.close();

    throw new ServletException(pe1.getMessage());
      }

      HashMap servicesAndOptions = paosHeader.getServicesAndOptions();

      Set services = servicesAndOptions.keySet();

      String thisURL = req.getRequestURL().toString();
      String[] queryItems = { "/IDPP/Demographics/Birthday" };
      PAOSRequest paosReq = null;
      try {
    paosReq = new PAOSRequest(thisURL,
                  (String)(services.iterator().next()),
                  thisURL,
                  queryItems);
      } catch (PAOSException pe2) {
    pe2.printStackTrace();
    throw new ServletException(pe2.getMessage());
      }
      System.out.println("PAOS request to User Agent side ---------------->");
      System.out.println(paosReq.toString());
      paosReq.send(res, true);
  }
```

EXAMPLE 8–1 PAOS Client Servlet From PAOS Sample *(Continued)*

```
  public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

      PAOSResponse paosRes = null;
      try {
    paosRes = new PAOSResponse(req);
      } catch (PAOSException pe) {
    pe.printStackTrace();
    throw new ServletException(pe.getMessage());
      }

      System.out.println("PAOS response from User Agent side -------------->");
      System.out.println(paosRes.toString());

      System.out.println("Data output after parsing -------------->");

      String dataStr = null;
      try {
    dataStr = paosRes.getPPResponseStr();
      } catch (PAOSException paose) {
    paose.printStackTrace();
    throw new ServletException(paose.getMessage());
      }
      System.out.println(dataStr);

      String msg = "Got the data: \\n" + dataStr;

      res.setContentType("text/plain");
      res.setContentLength(1+msg.length());

      PrintWriter out = new PrintWriter(res.getOutputStream());

      out.println(msg);

      out.close();
  }
}
```

◆ ◆ ◆ CHAPTER 9

Reading and Writing Log Records

Sun Java™ System Federated Access Manager provides a Logging Service for recording information such as user activity, traffic patterns, and authorization violations. This chapter contains information on how to implement and customize the logging functionality. It contains the following sections:

About the Logging Service

The Logging Service extracts information from the principal's session data structure and writes it to the configured log format — either a flat file or a relational database — when processing a request for logging. This information might include access denials and approvals, authentication events, and authorization violations. Administrators can use the logs to track user actions, analyze traffic patterns, audit system usage, review authorization violations, and troubleshoot. Logged information is recorded in a centralized directory; by default, `/fam/fam/log`. For more information on user sessions and the session data structure see Chapter 5, "User Session and Single Sign-On Processes," in *Sun Federated Access Manager 8.0 Technical Overview*. For information on how the Logging Service works, see Chapter 11, "Logging and the Java Enterprise System Monitoring Framework," in *Sun Federated Access Manager 8.0 Technical Overview*.

Using the Logging Interfaces

The Logging Service contains application programming interfaces (API) and service provider interfaces (SPI). The Logging API are used to add the logging functionality to a client application while the SPI can be used to develop custom plug-ins to add functionality to the Logging Service. The following sections contain information on these interfaces.

- "Implementing Logging with the Logging API" on page 194
- "Developing Plug-ins with the Logging SPI" on page 198

Implementing Logging with the Logging API

The Logging Service API provide the tools for all Federated Access Manager internal services and remote applications running the Client SDK to create, retrieve, submit, or delete log records. The API are contained in the `com.sun.identity.log` package.

Note – The Logging Service API extend the core logging API in the Java Platform, Standard Edition Development Kit (JDK). For more information, see Java SE Reference at a Glance.

The following sections have more information.

- "Writing Log Records" on page 194
- "Reading Log Records" on page 196

For more information see the *Federated Access Manager 8.0 Java API Reference*.

Writing Log Records

In writing log records, the Logging Service verifies that the logging requester has the proper authority to log, then writes the information to the configured storage medium, formatting and completing the record's columns. Applications make logging calls using the `getLogger()` method which returns a `Logger` object. Each `Logger` keeps track of a log level and discards log requests that are below this level. (There is one `Logger` object per log file.) The `Logger` object allocates a `LogRecord` which is written to the log file using the `log()` method. An `ssoToken`, representing the user's session data, is passed to the `LogRecord` constructor and used to populate the appropriate fields to be logged. Federated Access Manager contains plug-ins to write log records in the following ways:

- Writing to the host's flat file system
- Writing to the host's flat file system with added signing of the `LogRecord` and periodic verification
- Writing to a relational database
- Writing to a remote instance of Federated Access Manager

Note – The Logging Service requires two session tokens. Creating the `LogRecord` requires an `ssoToken` for the subject about whom the `LogRecord` is being written. Writing the `LogRecord` requires an `ssoToken` for the entity requesting the logging of the record.

The following parameters may have values logged when the `addLogInfo()` method is invoked. All columns except for *time*, *Data*, and *NameID* may be selected for exclusion from the record written to the log file.

time — The date and time is retrieved from Federated Access Manager and added by the Logging Service.

Data — The event being logged as defined in the `message` string specified in the `LogRecord()` constructor call.

ModuleName — The value specified for the `LogConstants.MODULE_NAME` property in the `addLogInfo()` call. For example, the RADIUS module might be specified in an authentication attempt.

Note – If no value is specified, this field will be logged as *Not Available*.

MessageID — The value specified for the `LogConstants.MESSAGE_ID` property in an `addLogInfo()` call.

Note – If no value is specified, this field will be logged as *Not Available*.

Domain — The value for this field is extracted from the `SSOToken` and corresponds to either the subject userID's domain, or organization.

ContextID — The value for this field is extracted from the `SSOToken` and corresponds to the subject userID's session context.

LogLevel — The logging level, passed to the `LogRecord()` constructor, at which this record is being logged.

LoginID — The value for this field is extracted from the `SSOToken` and corresponds to the subject userID's Principal name.

NameID — The value specified for the `LogConstants.NAME_ID` property in an `addLogInfo()` call. It is an alias that maps to the actual userID.

Note – If no value is specified, this field will be logged as *Not Available*.

IPAddr	The value for this field is extracted from the `SSOToken` and corresponds to the originating point of the action being logged.
LoggedBy	The identifier in this field is extracted from the logging requestor's `SSOToken` specified in the `Logger.log()` call.
HostName	The host name corresponding to the originating point of the action being logged is derived from the *IPAddr* in the user's `SSOToken`, if it can be resolved.

Note – Resolving host names is disabled by default; enable this feature by toggling the Log Record Resolve Host Name system configuration attribute under Logging Service. If disabled, the *HostName* value is taken from the user's `SSOToken` and the `IPAddr` value is logged as *Not Available*.

Reading Log Records

When handling log reading requests, a valid `SSOToken` must be provided. The Logging Service verifies that the requester has the proper authority after which it retrieves the requested records from the configured storage medium. The `LogReader` class provides the mechanism to read a log file and return the appropriate data to the caller. It provides the authorization check, reads the data, applies the query (if any), and returns the result as a string. The `LogQuery` is constructed using the `getLogQuery()` method.

Note – Unless all records from a log file are to be retrieved, at least one `LogQuery` must be constructed. The `LogQuery` objects qualify the search criteria.

A `LogQuery` may specify a list of `QueryElements`, each containing a value for a field (column) and a relationship. `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

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 Federated Access Manager resource limits (including those of the hosting system) may be exceeded.

The following sample code queries for all successful authentications in domain `dc=sun,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=sun,dc=com",
        QueryElement.EQ);
lq.addQuery(qe2);
```

In this code, assuming that `dc=sun,dc=com` is the root domain, changing the `qe2` relationship field to `QueryElement.EW` or `QueryElement.CN` changes the query to include all successful authentications in all domains. To read the example query from the `amAuthentication.access` log, assuming presence of an `SSOToken`, add the following:

```
 String[][] result = new String[1][1];
    result = read("amAuthentication.access", lq, ssoToken);
```

Note – The first record in a log (row 0) contains the field and column names.

Developing Plug-ins with the Logging SPI

The Logging SPI is contained in the `com.sun.identity.log.spi` package. The interfaces can be used for plugging in authorization support and an aspect related to secure logging. They can also be used as models to develop the plug-ins with customized features. The following sections have more information.

Provides an interface to define the actions that need to be taken depending on the return value of the Log Verification process.

- "Authorization" on page 198
- "Verification" on page 199

For more information, see the *Federated Access Manager 8.0 Java API Reference*.

Authorization

The Logging Service enables you to determine if an entity is authorized to perform the specified log operation (usually write or read). The `IAuthorizer` interface accepts an `SSOToken` and the `LogRecord`. The `Authorizer` class gets an instance of the class defined; by default, `com.sun.identity.log.spi.ISAuthorizer` in `amLogging.xml`. The determination is based on the authorization of the owner of the session token performing the event. There are several ways to accomplish this detemination. 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 Federated Access Manager, into this role.

2. **Instantiate a `PolicyEvaluator` using the following steps.**

   This entails defining a policy XML service file to model log access and registering it with Federated Access Manager.

   a. **Implement the `com.sun.identity.log.spi.IAuthorizer` interface with the desired functionality.**

   b. **Add the implementing class in the classpath of Federated Access Manager.**

   c. **Modify the property `iplanet-am-logging-authz-class` in the `amLogging.xml` file with the name of the new class.**

3 **Call** `PolicyEvaluator.isAllowed(ssotoken, logname).`

Verification

The `IVerifierOutput` interface defines the actions that need to be taken depending on the return value of the log verification process. If secure logging is enabled, the log files are checked periodically to detect any attempted tampering. If tampering is detected, the action taken can be customized using the following procedure.

▼ To Customize Actions to be Taken in Secure Logging

1 **Implement the** `IVerifierOutput` **interface with the desired functionality.**

2 **Add the implementing class to the classpath of Federated Access Manager.**

3 **Modify the** `iplanet-am-logging-verifier-action-class` **property in the** `amLogging.xml` **file with the name of the new class.**

Logging to a Second Instance of Federated Access Manager

For a remote instance of Federated Access Manager to use a second instance's Logging Service, set the Logging Service URL in the remote instance's Naming Service to the URL of the instance of Federated Access Manager that will be performing the actual logging. The URL should be in the following form:

`http://`*host*`:`*port*`/fam/loggingservice`

Note – There is no interface to read logs from a server remote to Federated Access Manager.

Implementing Remote Logging

Federated Access Manager supports remote logging. If your remote application is running in a container such as Sun Java System Application Server or Sun Java System Web Server, run the following commands to set the applicable 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
```

Note – The `-Djava.util.logging.manager` property is set in the `server.xml` file of the Web Server. Other JVM options are typically added to the `server.xml` file in Web Server, or to the `domain.xml` file in Application Server.

You must also 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 the following.

- If the application can execute in the either Java Virtual Machine (JVM) local to the instance of Federated Access Manager to which you are logging, or in a JVM remote to the instance of Federated Access Manager to which you are logging, see "If Client Executes in Local or Remote JVM" on page 200. The configuration also involves whether or not you want the Federated Access Manager `LogManager` class to override the native `LogManager` class.
- If the application can execute only in a JVM remote to the instance of Federated Access Manager to which you are logging, see "If Client Executes in Remote JVM Only" on page 201. The configuration also involves whether or not you want the Federated Access Manager `LogManager` class to override the native `LogManager` class.
- If SSL is enabled and uses JSS for Federated Access Manager, see "If SSL is Enabled" on page 202.

If Client Executes in Local or Remote JVM

When the client application can execute in either the local Federated Access Manager JVM or a remote JVM, choose one of the following configurations:

- If it is acceptable for the native `LogManager` class to be overridden by the Federated Access Manager `LogManager` class in the JDK1.4 environment, set the following variables:

  ```
  -D"java.util.logging.manager=com.sun.identity.log.LogManager"
              s1is.LogConfigReader"
  ```

- If it is *not* acceptable for the native `LogManager` class to be overridden by the Federated Access Manager `LogManager` class in the JDK1.4 environment, set the following variables:

  ```
  -DLOG_COMPATMODE=Off
  -Ds1is.java.util.logging.config.class=com.sun.identity.log.
              s1is.LogConfigReader
  ```

If Client Executes in Remote JVM Only

When the client application can execute only in a remote JVM, choose one of the following configurations:

- If it is acceptable for the native `LogManager` class to be overridden by the Federated Access Manager `LogManager` class in the JDK1.4 environment, 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 the 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 Federated Access Manager `LogManager` class in the JDK1.4 environment, follow these steps:
  1. Set the following variables:

     ```
     -DLOG_COMPATMODE=Off

     -Ds1is.java.util.logging.config.file=
     /AccessManager-base/SUNwam/lib/LogConfig.properties
     ```

  2. In `LogConfig.properties`, or in the `logging.properties` file supplied by the 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 enabled and uses JSS for Federated Access Manager, set the following parameter:

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

Logging Samples

Federated Access Manager provides two comprehensive sample logging programs in the *path-to-context-root*`/fam/samples/logging` directory. `LogSample.java` is a log-writing program, and `LogReaderSample.java` is a log-reading program.

- "`LogSample.java`" on page 202
- "`LogReaderSample.java`" on page 202

`LogSample.java`

`LogSample.java` authenticates a user with Federated Access Manager, creates a `LogRecord`, and writes the record to the specified log. The configuration of the Logging Service determines whether the log records go to a flat file or to a relational database. This sample is part of the Client SDK. More information can be found in Chapter 1, "Enhancing Remote Applications Using the Client Software Development Kit."

`LogReaderSample.java`

Remark 9–1 Reviewer **Not sure where this sample is so I haven't rewritten this info. Needs more info on sample.**

`LogReaderSample.java` requires three command-line arguments which are used to authenticate with Federated Access Manager. 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 the 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.

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)`

```
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.

Using the Logging Sample Files

Remark 9–2 Reviewer **Still valid? This info is on Client SDK chapter also.**

The sample files demonstrate how you can use the Federated 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" on page 206
- "To Run the Sample Programs on Windows 2000" on page 208

▼ 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 where Federated Access Manager 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, else set it to `false`
`LOCAL_LOGGING`	Set this variable to `true` if you are executing this sample at complete Federated Access Manager installation which will perform local logging. If you are executing this sample from a `SUNWamsdk` only install 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, Federated 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 Federated Access Manager 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 Federated Access Manager installation which will perform local logging. If you are executing this sample from an `SUNWamsdk` only install, 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, Federated 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
```

◆ ◆ ◆ CHAPTER 10

Securing Web Services

Web services are developed using open standards such as XML, SOAP, WSDL and HTTPS. Sun Java™ System Federated Access Manager provides the functionality to secure web services communications using authentication agents and the Security Token Service. This chapter contains the following sections:

About Web Services Security

A *web service* is an application whose functionality and interfaces are exposed through open technology standards including the eXtensible Markup Language (XML), SOAP, the Web Service Description Language (WSDL) and HTTP(S). A web service client (WSC) accesses a web service provider (WSP) by sending a SOAP message to a service endpoint identified by a URI; after receiving the request, the WSP responds appropriately with a SOAP response. The built-in openness of these technologies though creates security risks. Initially, securing these web services communications was addressed using *transport level security* in which the complete message was encrypted and transmitted using Secure Sockets Layer (SSL) with mutual authentication. But with current enterprise topologies (including proxies, load balancers, data centers, and the like) security must be addressed when intermediaries are involved. Web services must be prepared to:

- Pass fine-grained security data (for example, identity attributes for authorization).
- Enable one or more trusted authorities to broker trust between communicating entities.

- Maintain security on a per message basis.
- Maintain transport layer independence.

These requirements call for *message level security* (also referred to as *application level security* and *end-to-end security*) in which only the content of the message is encrypted. Message level security embeds all required security information in a message's SOAP header. Additionally, encryption and digital signatures can be applied to the data itself. The advantages of message level security are that:

- Security stays with the message through all intermediaries, across domain boundaries, and after the message arrives at its destination.
- Security can be selectively applied to different portions of the message.
- Security is independent of the application environment and transport protocol.

To address message level security in web services communications, organizations such as the Organization for Advancement of Structured Information Standards (OASIS), the Liberty Alliance Project and the Java Community Process (JCP) have proposed specifications based on open standards and from them Federated Access Manager has implemented "Authentication Agents" on page 212 and "The Security Token Service" on page 218.

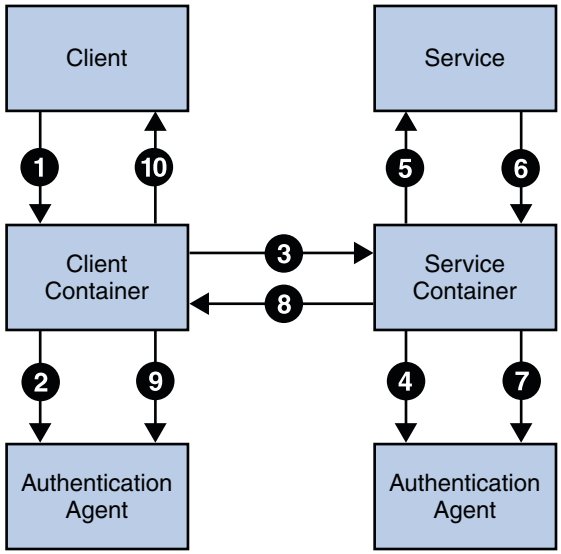
Authentication Agents

The Java Community Process (JCP) primarily guides the development and approval of Java technical specifications, one of which is the Java Specification Request (JSR) 196. JSR 196 is a draft of the *Java Authentication Service Provider Interface for Containers*. It defines a standard service provider interface (SPI) with which an authentication agent can be developed to police Java EE containers on either the client side or the server side. These agents may establish the authenticated identities used by the containers allowing:

- A server side agent to verify security tokens or signatures on incoming requests and extract principal data or assertions before adding them to the client security context.
- A client side agent to add security tokens to outgoing requests, sign messages, and interact with the trusted authority to locate targeted web service providers.

Note – The JSR 196 draft specifications are available at `http://www.jcp.org/en/jsr/detail?id=196`.

A typical interaction between a WSC and a WSP begins with a request from the WSC. The container on which the WSP is deployed receives the request and dispatches it to perform the requested operation. When the web service completes the operation, it creates a response that is returned back to the client. The following illustration and procedure illustrates a scenario when both client and service web containers employ the Java Authentication SPI.



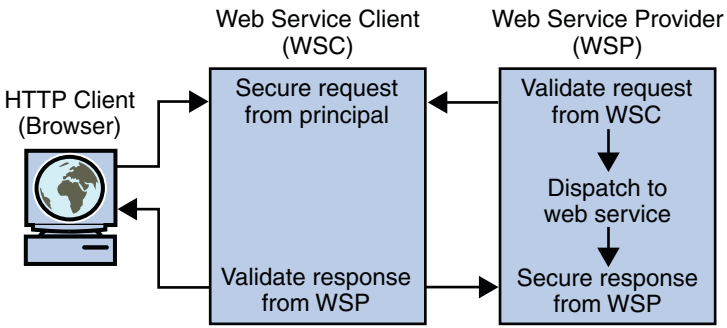
1. The client browser's attempt to invoke a web service is intercepted by the client's web container.
2. The deployed authentication agent on the client's web container is invoked to secure the request (based on the security policy of the web service being invoked).
3. The client's web container sends the secured request message to the web service.
4. The web service's web container receives the secured request message and it's deployed authentication agent is invoked to validate the request and obtain the identity of the caller.
5. Assuming successful authentication, the web service's web container invokes the requested web service.
6. This action (the invocation of the web service) is returned to the web service's web container as a response.
7. The deployed authentication agent on the web service's web container is invoked to secure the response message.
8. The web service's web container sends the secured response message to the client.
9. The deployed authentication agent on the client's web container is invoked to validate the secured response message.
10. The invocation of the web service is returned to the client browser.

Security processes can be delegated to an authentication agent at any of the following interaction points.

- Securing a request on the client side

- Validating a request on the provider side
- Securing a response on the provider side
- Validating a response on the client side

Thus, when a WSC and a WSP are both deployed in a Java EE web container protected by an authentication agent, the initial request from the WSC is intercepted by the authentication agent on the client side. The client side agent queries a *trusted authority* (for example, the Security Token Service) to retrieve the necessary authorization credentials and secure them to the request. The request is then passed to the WSP. The authentication agent on the provider side receives the request to validate the authorization credentials. If validation is successful, the request is exposed to the web service and a response is created using the sender's credentials and the application specific request. The response is then intercepted by the authentication agent on the provider side to secure it and return it to the WSC. Upon receiving the response, the authentication agent on the client side validates it and dispatches it to the client application. This is illustrated in the following illustration.



This authentication agent uses an instance of Federated Access Manager for all authentication decisions. Web services requests and responses are passed to the authentication modules using standard Java representations based on the transmission protocol. Currently, the following agents are provided.

- "HTTP Authentication Agent" on page 214
- "SOAP Authentication Agent" on page 216

HTTP Authentication Agent

The HTTP authentication agent protects the endpoints of a web service that uses HTTP for communication. After the HTTP authentication agent is deployed in a web container on the WSP side, all HTTP requests for access to web services protected by the agent are redirected to the login and authentication URLs defined in the Federated Access Manager configuration data store on the WSC side. The configurable properties are:

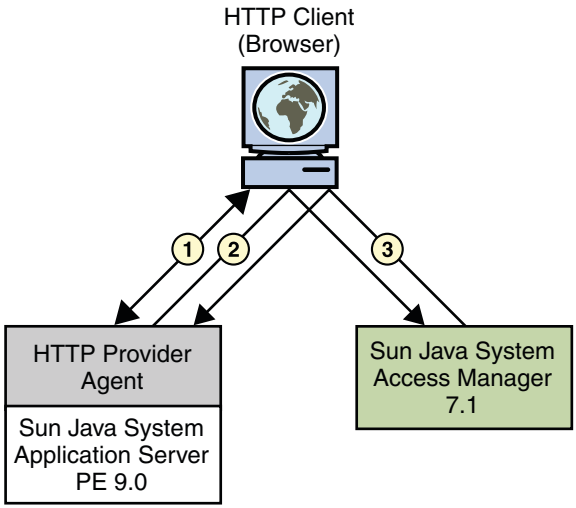
- **com.sun.identity.loginurl=***fam_protocol*://*fam_host*:*fam_port*/fam/UI/Login**
- **com.sun.identity.liberty.authnsvc.url=***fam_protocol*://*fam_host*:*fam_port*/fam/Liberty/a**

Note – Application Server 9 has the ability to configure only one HTTP agent per instance. Therefore, all authentication requests for all web applications hosted in the container will be forwarded to the one configured agent.

When the WSC makes a request to access a web application protected by an HTTP authentication agent (**1** in the illustration below), the agent intercepts the request and redirects it (via the browser) to Federated Access Manager for authentication (**2**). Upon successful authentication, a response is returned to the application, carrying a token as part of the Java EE Subject (**3**). This token is used to bootstrap the appropriate Liberty ID-WSF security profile. If the response is successfully authenticated, the request is granted (**3**).

Note – For this release, the HTTP authentication agent is used primarily for bootstrapping. Future releases will contain information on how to protect web applications.

The following figure illustrates the interactions described.



Note – The functionality of the HTTP Provider agent is similar in to that of the Java EE policy agents when used in SSO ONLY mode. This is a non restrictive mode that uses only the Federated Access Manager Authentication Service to authenticate users attempting access. For more information on Java EE policy agents, see the *Sun Java System Access Manager Policy Agent 2.2 User's Guide*.

SOAP Authentication Agent

The SOAP authentication agent secures SOAP messages between a WSC and a WSP. The agent can be configured for use as an authentication provider on either the WSC server or the WSP server. This initial release encapsulates the Liberty Identity Web Services Framework (Liberty ID-WSF) SOAP Binding Specification as implemented by Access Manager and supports the following:

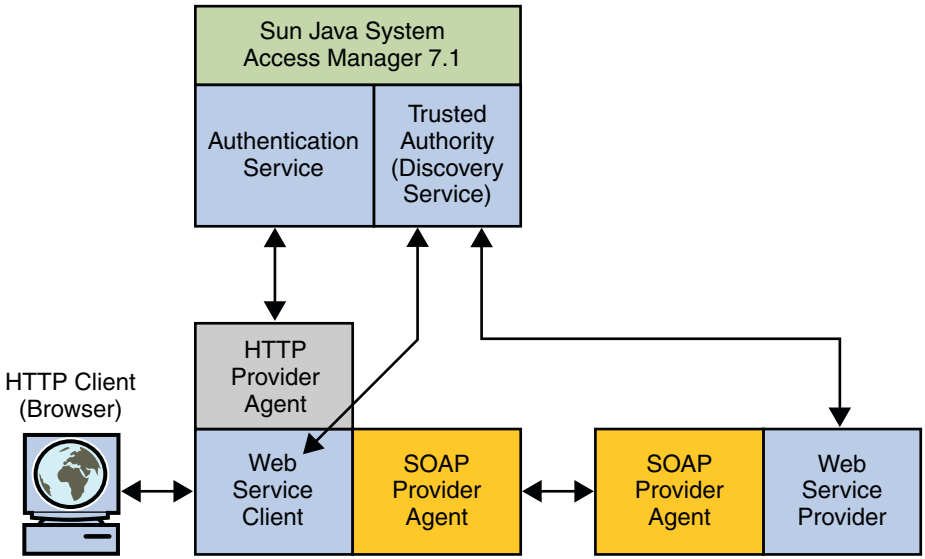
- "Supported Liberty Alliance Project Security Tokens" on page 216
- "Supported Web Services-Interoperability Basic Security Profile Security Tokens" on page 217

Note – The configuration process for the SOAP authentication agent is described in Installing the Policy Agent 2.2 for Sun Java System Application Server 9.0 / Web Services.

Supported Liberty Alliance Project Security Tokens

In a scenario where security is enabled using Liberty Alliance Project tokens, the HTTP client requests (via the WSC) access to a service. The HTTP authentication agent redirects the request to the Access Manager Authentication Service for authentication and to determine the security mechanism registered by the WSP and obtain the security tokens expected. After a successful authentication, the WSC provides a SOAP body while the SOAP authentication agent on the WSC side inserts the security header and a token. The message is then signed before the request is sent to the WSP.

When received by the SOAP authentication agent on the WSP side, the signature and security token in the SOAP request are verified before forwarding the request on to the WSP itself. The WSP then processes it and returns a response, signed by the SOAP authentication agent on the WSP side, back to the WSC. The SOAP authentication agent on the WSC side then verifies the signature before forwarding the response on to the WSC. The following diagram illustrates the interactions as described.



The following Liberty Alliance Project security tokens are supported in this release:

X.509 A secure web service uses a PKI (public key infrastructure) in which the web service consumer supplies a public key as the means for identifying the requester and accomplishing authentication with the web service provider. Authentication with the web service provider using processing rules defined by the Liberty Alliance Project.

BearerToken A secure web service uses the Security Assertion Markup Language (SAML) *SAML Bearer token* confirmation method. The web service consumer supplies a SAML assertion with public key information as the means for authenticating the requester to the web service provider. A second signature binds the assertion to the SOAP message This is accomplished using processing rules defined by the Liberty Alliance Project

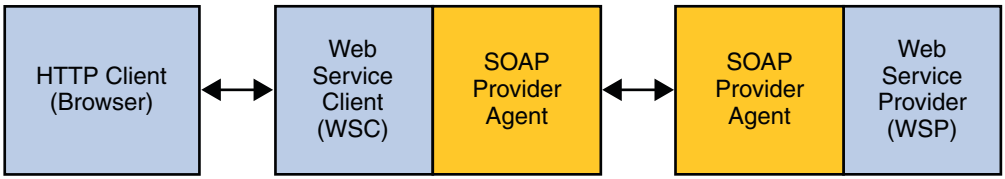
SAMLToken A secure web service uses the SAML *holder-of-key* confirmation method. The web service consumer adds a SAML assertion and a digital signature to a SOAP header. A sender certificate or public key is also provided with the signature. This is accomplished using processing rules defined by the Liberty Alliance Project.

Supported Web Services-Interoperability Basic Security Profile Security Tokens

In a scenario where security is enabled using Web Services-Interoperability Basic Security Profile (WS-I BSP) tokens, the HTTP client requests (via the WSC) access to a service. The SOAP authentication agent redirects the request to the Access Manager Authentication Service for authentication and to determine the security mechanism registered by the WSP and obtain the expected security tokens. After a successful authentication, the WSC provides a SOAP body

while the SOAP authentication agent on the WSC side inserts the security header and a token. The message is then signed before the request is sent to the WSP.

When received by the SOAP authentication agent on the WSP side, the signature and security token in the SOAP request are verified before forwarding the request on to the WSP itself. The WSP then processes it and returns a response, signed by the SOAP authentication agent on the WSP side, back to the WSC. The SOAP authentication agent on the WSC side then verifies the signature before forwarding the response on to the WSC. The following diagram illustrates the interactions as described.



The following WS-I BSP security tokens are supported in this release.

User Name A secure web service requires a user name, password and, optionally, a signed the request. The web service consumer supplies a username token as the means for identifying the requester and a password, shared secret, or password equivalent to authenticate the identity to the web service provider.

X.509 A secure web service uses a PKI (public key infrastructure) in which the web service consumer supplies a public key as the means for identifying the requester and accomplishing authentication with to the web service provider.

SAML-Holder-Of-Key A secure web service uses the SAML *holder-of-key* confirmation method. The web service consumer supplies a SAML assertion with public key information as the means for authenticating the requester to the web service provider. A second signature binds the assertion to the SOAP payload.

SAML-SenderVouches A secure web service uses the SAML *sender-vouches* confirmation method. The web service consumer adds a SAML assertion and a digital signature to a SOAP header. A sender certificate or public key is also provided with the signature.

The Security Token Service

When a WSC communicates with a WSP it must first connect with a trusted authority to determine the security mechanism and, optionally, obtain the security token expected by the WSP. This information is registered with the trusted authority by the WSP. The Security Token Service is a trusted authority that provides issuance and management of security tokens; that is, it makes security statements or claims often, although not required to be, in cryptographically protected sets. The Federated Access Manager trust brokering process is as follows.

1. An authenticated WSC requests a token to access a particular WSP.

2. The Security Token Service verifies the credentials presented by the WSC.
3. In response to an affirmative verification, the Security Token Service issues a security token that provides proof that the client has been authenticated.
4. The WSC presents the security token to the WSP.
5. The WSP verifies that the token was issued by a trusted Security Token Service, affirming authentication.

The Security Token Service issues, renews, cancels, and validates security tokens that can contain an identifier for either the WSC or the actual end user. It also allows you to write a proprietary token providers using the included service provider interfaces (SPI). Finally, it provides application programming interfaces (API), based on the WS-Trust protocol, that allow applications to access the service. The WS-Trust protocol defines the formats of the messages used to request security tokens and the responses to those messages as well as mechanisms for key exchange. By default, the Security Token Service serves the following tokens:

User Name Token Carries basic information (username and, optionally, a password or shared secret) for purposes of authenticating the user identity to the WSP. Communication is done in plain text so SSL over HTTPS transport must be used to protect the credentials.

X.509 Token Contains an X.509 formatted certificate for authentication using credentials created with a public key infrastructure (PKI). In this case, the WSC and WSP must trust each other's public keys or share a common, trusted certificate authority.

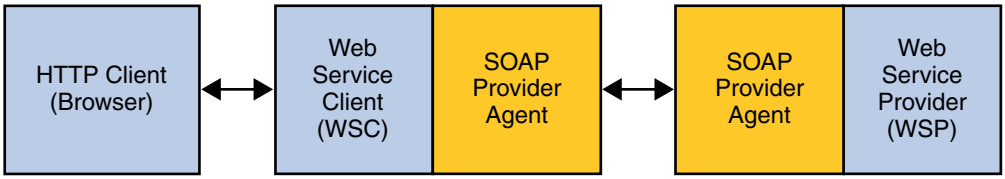
SAML-Holder-Of-Key Token Uses the SAML *holder-of-key* confirmation method whereby the WSC supplies a SAML assertion with public key information as the means for authenticating the requester to the web service provider. A second signature binds the assertion to the SOAP payload.

SAML-SenderVouches Token Uses the SAML *sender-vouches* confirmation method whereby the WSC adds a SAML assertion and a digital signature to a SOAP header. A sender certificate or public key is also provided with the signature.

Note – The Security Token Service issues security tokens allowed by the WS-I Basic Security Profile while the Federated Access Manager Discovery Service issues tokens based on the Liberty Alliance Project specifications; thus, the two services are independent. See "Discovery Service" on page 175 for more information on the latter.

In a scenario where security is enabled using Web Services-Interoperability Basic Security Profile (WS-I BSP) tokens, the HTTP client requests (via the WSC) access to a service. The SOAP authentication agent redirects the request to the Access Manager Authentication Service for authentication and to determine the security mechanism registered by the WSP and obtain the expected security tokens. After a successful authentication, the WSC provides a SOAP body while the SOAP authentication agent on the WSC side inserts the security header and a token. The message is then signed before the request is sent to the WSP.

When received by the SOAP authentication agent on the WSP side, the signature and security token in the SOAP request are verified before forwarding the request on to the WSP itself. The WSP then processes it and returns a response, signed by the SOAP authentication agent on the WSP side, back to the WSC. The SOAP authentication agent on the WSC side then verifies the signature before forwarding the response on to the WSC. The following diagram illustrates the interactions as described.



Accessing the Security Token Service

API : com.sun.identity.wss.sts

Extending the Security Token Service

SPI : com.sun.identity.wss.security

Configuring the Security Token Service

How does WSC know to talk to STS ? > WSP WSDL (retrieved via MEX) > Defines configuration at WSC provider to choose STS issued token and selects the STS Agent (STS client) to retrieve STS service end points.

For setting up client to talk to FAM STS services (STS end points), please deploy fam-client-jdk15.war and follow sts/index.html under this deployment. You need to deploy this client war on App server. Also before this, on same App server, setup security modules (based on JSR 196) for STS security, using ../products/wssagents/built/dist/openssowssproviders.zip (unzip and follow README).

Testing Web Services Security

Access Manager Policy Agent 2.2 for Sun Java System Application Server 9.0 / Web Services is installed with Sun Microsystems' best practice applications called Java BluePrints. The Java BluePrints program defines the application programming model for the Java Enterprise Edition (Java EE) platform. The following sections describe the included BluePrints which focus on web services security.

- "Stock Service Sample" on page 221
- "Calendar Service Sample" on page 221

Stock Service Sample

This BluePrint focuses on building a web service provider (WSP) and a web service client (WSC), authenticating the WSC before access to the service is given, and guaranteeing the integrity of the authentication data. This is accomplished by using Web Services Interoperability Basic Security Profile (WS-I BSP) tokens to secure communications between the participants. The BluePrint encompasses a web service that provides details for a given stock symbol. The instructions for the Stock Service BluePrint is the `index.html` file found in */javaee.home*`/blueprints/ws-security/stock-jaxrpc/` directory.

Calendar Service Sample

This BluePrint focuses on securing an identity-based WSP. Identity-based web services must know the identity of the user accessing the service. The Calendar Service BluePrint is a calendar service which uses the identity of the user to enforce permission checks on the event(s) being accessed. In securing an identity-based WSP, the identity accessing the service (via the WSC) is authenticated before being given access. Additionally, the WSC would also be authenticated by the WSP before being given access. The instructions for the Calendar Service BluePrint is the `index.html` file found in */javaee.home*`/blueprints/ws-security/calendar-jaxrpc/` directory.

Keystores

J2EE agents work with Access Manager to protect resources. However, for security purposes, these two pieces of software can only interact with each other after the J2EE agent authenticates with Access Manager by supplying an agent profile name and password. The agent profiles we are using (`wscWSC`, `LibertyBearerToken`, etc.) are configured to use the following keystores, by default:

- The Access Manager Policy Agent 2.2 for Application Server 9.0 / Web Services uses the client keystore shipped with the Java EE SDK, `amclientkeystore.jks`, as its default client keystore. It is located in *javaee.home*`/addons/accessmanager` for installations of Java

Application Platform SDK (when Download or Download with JDK is selected), and in *javaee.home*/`addons/amserver` for installations of Java Application Platform SDK or Java EE 5 SDK Update 1 (when Download with Tools is selected), and NetBeans Enterprise Pack 5.5

- The single WAR instance of Access Manager uses the server keystore shipped with the Java EE SDK, `keystore.jks`, for its default keystore. It is located in *javaee.home*/`domains`/*domain_name*/`config/amflatfiledir/amserver` for installations of Java Application Platform SDK (all downloads), Java EE 5 SDK Update 1 (only when Download with Tools is selected), and NetBeans Enterprise Pack 5.5.

You can configure for a custom keystore, though. The following procedure describes the necessary steps.

Note – For more information on agent profiles, see "Agents Profile" in *Sun Java System Access Manager 7.1 Administration Guide* in the *Sun Java System Access Manager 7.1 Administration Guide*

During the installation of the J2EE agent, you must provide a valid agent profile name and the respective password to enable authentication attempts to succeed.

▼ To Configure for a Custom Keystore

1 **Export the certificate for the alias `amserver` using the following command:**

`keytool -list -keystore` *keystore_file* **`-alias amserver -rfc`**

2 **Store the exported X509 certificate, using the RFC format, in a file named `server.txt`.**

3 **Export the certificate from your custom keystore using the following command:**

`keytool -list -keystore` *custom_keystore_file* **`-alias`** *key alias* **`-rfc`**

key alias is the alias of the private key used by the WSC to sign SOAP messages.

4 **Store the exported X509 certificate, using the RFC format, in a file named `client.txt`.**

5 **Import the stored `amserver` certificate into the agent's custom keystore file using the following command:**

`keytool -import -keystore` *custom_keystore_file* **`-alias`** *custom_alias* **`-file server.txt`**

6 **Import the stored custom keystore's certificate into the Access Manager keystore file using the following command:**

`keytool -import -keystore` *custom_keystore_file* **`-alias`** *custom_alias* **`-file client.txt`**

7 Generate a Discovery Service token for the WSC that will use the custom keystore with the following command:

`keytool -import -keystore` *custom_keystore.jks* **`-alias amserver -file server.txt`**

This allows the WSP which uses the custom keystore to trust the Access Manager Discovery Service.

8 Edit the following properties in the client's `AMConfig.properties`:

- **`com.sun.identity.liberty.ws.wsc.certalias=`***alias_of_private_key_in_custom_client_keystore*

  This certificate is used by the Liberty X509/SAML profiles for signing the SOAP messages.
- **`com.sun.identity.liberty.ws.trustedca.certaliases=`***alias_of_private_key_in_custom_serve*

`AMConfig.properties` is located in *javaee.home*`/domains/`*domain_name*`/config` when the Java Platform, Enterprise Edition (Java EE) 5 SDK is installed and in *javaee.home*`/addons/amserver` when the Java EE 5 Tools Bundle is installed.

◆ ◆ ◆ CHAPTER 11

Identifying the Client Type

The Sun Java™ System Federated Access Manager 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, Federated 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:

About the Client Detection Service

The Federated 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. The client detection API are used to determine the protocol of the requesting client browser and retrieve the correctly formatted pages for the particular client type.

Caution – The Federated Access Manager console though cannot be accessed from any client type except HTML.

Since any user requesting access to Federated 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 Federated 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 hold 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 the configuration data store 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 Federated 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, Federated Access Manager will be accessed from a HTML browser.) The preferred way to enable the Client Detection Service is to use the Federated 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.

▼ To Enable Client Detection

1. **Import client data XML file using the `amadmin` command** *//FederatedAccessManager-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 Federated Access Manager.**

3 **Login to the Federated 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 the Federated 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 *//FederatedAccessManager-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>
```

Defining Client Data

In order to detect client types, Federated 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 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 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`	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, Federated Access Manager console does not. Therefore, enabling this function will not allow logging in to the console. This feature is provided for wireless applications and others that will support it.

`CcppAccept-Charset`	Defines the character encoding used by Federated 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, it 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`

Note – 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.

Using the Client Detection Interfaces

Federated 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 12

Using the Access Manager Utilities

Sun Java™ System Federated Access Manager 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 */FederatedAccessManager-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 */FederatedAccessManager-base*`/SUNWam/docs/` directory into the */FederatedAccessManager-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 Federated Access Manager installation. They are located in */FederatedAccessManager-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 */FederatedAccessManager-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 */FederatedAccessManager-base*`/SUNWam/samples/console/PasswordGenerator`.

◆ ◆ ◆ CHAPTER 13

The Federated Access Manager Notification Service

The Sun™ Java System Federated Access Manager 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 Federated 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 Federated 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 Federated 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 Federated 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 Federated 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 Federated 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 Federated 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 14

Updating and Redeploying Federated Access Manager WAR Files

Federated Access Manager contains a number of web archive (WAR) files. These packages contain Java servlets and JavaServer Pages™ (JSP) you can modify to customize Federated Access Manager to meet your needs. The chapter contains the following sections:

WAR Files in J2EE Software Development

Federated 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. JSP 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

WAR Files in Federated Access Manager

When you customize Federated Access Manager, you must also modify the Federated Access Manager WAR files. The modifications in turn result in changes to the web components.

Federated Access Manager provides two types of WAR files. One type of 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 Federated Access Manager. At installation, based on the source files in the staging

directory */FederatedAccessManager-base*/web-src/, both password.war and services.war are automatically generated and deployed into the */FederatedAccessManager-base*/SUNWam/war directory. When you want to customize Federated 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 Federated Access Manager, any customizations you have implemented may be overwritten.

The second type of Federated 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.war through the JES installer, or you can manually deploy one or both of them.

The following Access Manager WAR files are located in */FederatedAccessManager-base*/SUNWam directory:

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 */FederatedAccessManager-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 */FederatedAccessManager-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 */FederatedAccessManager-base*/web-src/services/* source files.

The following Access Manager WARs are located in the */FederatedAccessManager-base*/SUNWam/war directory:

.

am_server.war — Use this WAR to manually install Federated 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 Federated Access Manager server. For more information, see "Running the Client SDK Samples" on page 19.
`amauthdistui.war`	Use this WAR to manually install the Distributed Authentication UI server on a container remote from the Federated 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 279.
`amconsole.war`	Federated 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 */FederatedAccessManager-base*`/web-src/services/console`, when Access Manager is installed. You cannot customize this WAR.
`console.war`	Federated 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 */FederatedAccessManager-base*`/web-src/services/console`, when Access Manager is installed. You can customize this WAR. For more information, see Chapter 15, "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 */FederatedAccessManager-base*`/web-src/services/common`, when Federated Access Manager is installed. You cannot customize this WAR.
`password.war`	Federated 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 */FederatedAccessManager-base*`/web-src/services/password`, when Federated Access Manager is installed. You can customize this WAR. For more information, see the section "password.war" on page 243.
`services.war`	Federated Access Manager uses this WAR to build the UI for various services. The `services.war` file is automatically generated and deployed, based on the source code in */FederatedAccessManager-base*`/web-src/services/services`, when

Federated Access Manager is installed. You can customize this WAR. For more information, see the section "services.war" on page 243.

password.war

The `password.war` contains files used by the Federated 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 */FederatedAccessManager-base*`/SUNWam/web-src/password/WEB-INF/`.
- `JSPs` located in */FederatedAccessManager-base*`/SUNWam/web-src/password/password/ui/` .
- Image files located in */FederatedAccessManager-base*`/SUNWam/web-src/password/password/images/` .
- Stylesheets located in */FederatedAccessManager-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.

- `JARs` located in */FederatedAccessManager-base*`/SUNWam/web-src/password/WEB-INF/lib/`.
- Tag library descriptor (`.tld`) files located in */FederatedAccessManager-base*`/SUNWam/web-src/password/WEB-INF/`.

services.war

The `services.war` contains files used by various 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 */FederatedAccessManager-base*`/SUNWam/web-src/services/WEB-INF/`.
- JavaScript files are located in */FederatedAccessManager-base*`/SUNWam/web-src/services/js/`.

- JSP are located in the following directories:
  - */FederatedAccessManager-base*/SUNWam/web-src/services/config/auth/default/
  - */FederatedAccessManager-base*/SUNWam/web-src/services/config/federation/default/

  Image files are located in the following directories:
  - */FederatedAccessManager-base*/SUNWam/web-src/services/images/
  - */FederatedAccessManager-base*/SUNWam/web-src/services/fed_images/
  - */FederatedAccessManager-base*/SUNWam/web-src/services/login_images/

  Stylesheets are located in the following directories:
  - */FederatedAccessManager-base*/SUNWam/web-src/services/css/.
  - */FederatedAccessManager-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 Federated Access Manager to fail:

- Non-modifiable JARs are located in */FederatedAccessManager-base*/SUNWam/web-src/services/WEB-INF/lib/.
- Non-modifiable Tag Library Descriptor (.tld) files are located in */FederatedAccessManager-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 /FederatedAccessManager-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 */FederatedAccessManager-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 `WAR` must be redeployed to its 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 a `WAR` to be exploded. The servers themselves are deployed as a `WAR`. After WAR files are installed on these servers, you must restart all related servers.

- "Redeploying a Federated Access Manager WAR On BEA WebLogic Server 6.1" on page 245
- "Redeploying a Federated Access Manager WAR on Sun Java System Application Server 7.0" on page 246
- "Redeploying a Federated Access Manager WAR on IBM WebSphere Application Server" on page 247

Redeploying a Federated Access Manager 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 /FederatedAccessManager-base/SUNWam/WARnam
```

where the following variables are used:

protocol://server_host:server_port	The protocol `[http | https]` and fully-qualified name of the Federated 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`.
/FederatedAccessManager-base	The directory where Federated Access Manager 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 BEA WebLogic Server 6.1 documentation (`http://edocs.bea.com/wls/docs61/index.html`).

Redeploying a Federated Access Manager WAR 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
Federated 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//FederatedAccessManager-base
Application Server directory where Federated Access Manager 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 a Federated Access Manager WAR on IBM WebSphere Application Server

For detailed instructions on how to deploy a `WAR` in the IBM WebSphere Application Server, see the Application Server documentation.

◆ ◆ ◆ CHAPTER 15

Customizing the Administration Console

The Sun Federated Access Manager Administration Console is a web-based interface for creating, managing, and monitoring the identities, web services, and enforcement policies configured throughout a Federated Access Manager deployment. It is built with the 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 console's HTML pages.

This chapter describes the Administration Console, its pluggable architecture, and how to customize the Legacy mode user interface. It 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 Administration 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 Federated 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.

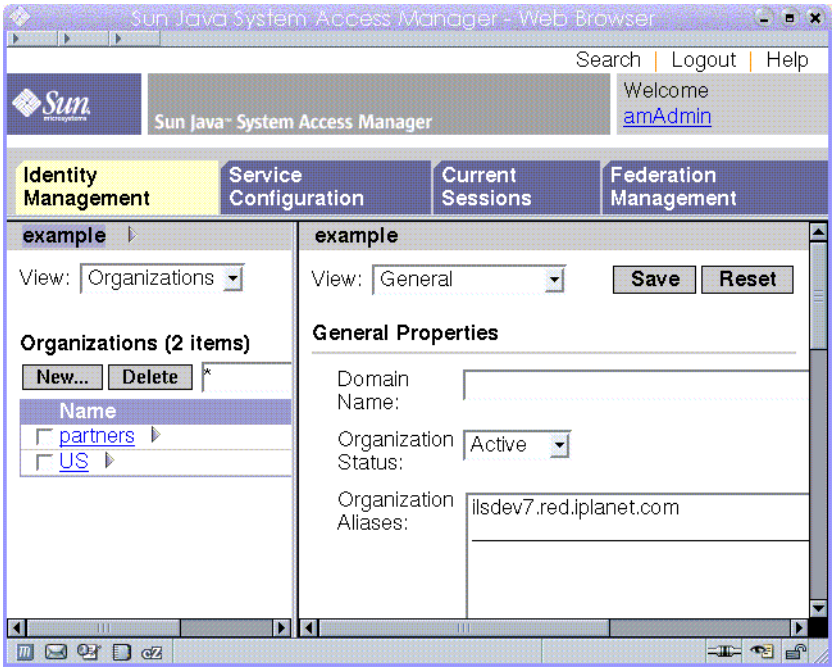


FIGURE 15–1 Legacy Mode Administration Console

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" on page 250
- "Plug-In Modules" on page 251
- "Accessing the Console" on page 251

Generating The Console Interface

When the Federated 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 Federated 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 Federated Access Manager. The URL used to access the Administration Console web application is:

```
http://AcceessManager-HostName.domain_name:port/
    amconsole
```

The first time the 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 Federated 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 */FederatedAccessManager-base*/SUNWam/web-src/services/console directory. The files in this directory provide the default 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 */FederatedAccessManager-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 documentation.

---

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 */FederatedAccessManager-base*/SUNWam/web-src/services/WEB-INF/.
- Modifiable JavaScript files are located in */FederatedAccessManager-base*/SUNWam/web-src/services/console/js/.
- Modifiable JSP are located in the following directories dependant upon the service that deploys them:
  - */FederatedAccessManager-base*/SUNWam/web-src/services/console/auth/

- */FederatedAccessManager-base*/SUNWam/web-src/services/console/federation/
- */FederatedAccessManager-base*/SUNWam/web-src/services/console/policy/
- */FederatedAccessManager-base*/SUNWam/web-src/services/console/service/
- */FederatedAccessManager-base*/SUNWam/web-src/services/console/session/
- */FederatedAccessManager-base*/SUNWam/web-src/services/console/user/

Modifiable image files are located in */FederatedAccessManager-base*/SUNWam/web-src/services/console/images/.

- Modifiable stylesheets are located in */FederatedAccessManager-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 */FederatedAccessManager-base*/SUNWam/web-src/services/WEB-INF/lib/.
- Tag Library Descriptor (`.tld`) files are located in */FederatedAccessManager-base*/SUNWam/web-src/services/WEB-INF/.

Creating Custom Organization Files

To customize the console for use by a specific organization, the */FederatedAccessManager-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 */FederatedAccessManager-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` */FederatedAccessManager-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 */FederatedAccessManager-base*/SUNWam/web-src/services/custom_directory/.

3 **Copy all the JSP files from the `console` directory into the new directory.**

/FederatedAccessManager-base/SUNWam/web-src/services/console contains the default JSP for Federated 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 */FederatedAccessManager-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 253.

```
<!--
 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 253.**

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 */FederatedAccessManager-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 */FederatedAccessManager-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 Federated Access Manager console.

To Modify The Service Configuration Display

A *service* is a group of attributes that are managed together by the Federated Access Manager console. Out-of-the-box, Federated 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 Federated Access Manager console, while code implementations within Federated Access Manager use the attribute values to run the service.

To Modify The User Profile View

The Federated 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://*Federated 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:
/FederatedAccessManager-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 */FederatedAccessManager-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 Federated Access Manager console.

To Customize Interface Colors

All the colors of the console are configurable using the Federated Access Manager style sheet `adminstyle.css` located in the */FederatedAccessManager-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 255 details the tag in `adminstyle.css`.

EXAMPLE 15–1 `BODY.navFrame` Portion of `adminstyle.css`

```
BODY.navFrame   {
                color: black;
                background: #ffffff;
```

EXAMPLE 15–1 `BODY.navFrame` Portion of `adminstyle.css` *(Continued)*

```
}
```

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 255 illustrates this concept.

EXAMPLE 15–2 `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 255 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 15–1 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
	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 */FederatedAccessManager-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 255 illustrates the default pairs in the file.

EXAMPLE 15–3 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 Federated 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 /`*FederatedAccessManager-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 Federated 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 */FederatedAccessManager-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 Federated Access Manager is installed on the Solaris™ operating system.

◆ ◆ ◆ CHAPTER 16

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 Federated Access Manager 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 263 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 16–1 Authentication User Interface Files and Their Locations at Installation

File Type	Default Location
"Staging Area for Files to be Customized" on page 264	*/FederatedAccessManager-base*/SUNWam/web-src/services
"Java Server Pages" on page 265	*/FederatedAccessManager-base*/SUNWam/web-src/services/config/auth/default
"XML Files" on page 267	*/FederatedAccessManager-base*/SUNWam/web-src/services/config/auth/default
"JavaScript Files" on page 270	*/FederatedAccessManager-base*/SUNWam/web-src/services/js
"Cascading Style Sheets" on page 271	</*FederatedAccessManager-base*/SUNWam/web-src/services/css
"Images" on page 271	*/FederatedAccessManager-base*/SUNWam/web-src/services/login_images
"Localization Files" on page 272	*/FederatedAccessManager-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 Federated Access Manager is installed, a staging area exists in the following location:

/FederatedAccessManager-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 Federated 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 277.

Java Server Pages

All authentication GUI pages are `.jsp` files with embedded JATO tags. You do not need to understand JATO to customize Federated 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:
/FederatedAccessManager-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 265 provides descriptions of templates you can customize. The templates are located in the following directory:

/FederatedAccessManager-base`/SUNWam/web-src/services/config/auth/default`

TABLE 16–2 Customizable JSP Templates

File Name	Purpose
`account_expired.jsp`	Informs the user that their account has expired and should contact the system administrator.

TABLE 16–2 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 16–2 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: */FederatedAccessManager-base*`/SUNWam/Auth_Module_Properties.dtd`. Federated 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:

/FederatedAccessManager-base`/SUNWam/web-src/services/config/auth/default` The table "XML Files" on page 267 provides descriptions of the authentication module configuration files.

XML files are looked up from the deployed location. In previous Federated Access Manager versions, the XML files were looked up from the installed location.

TABLE 16–3 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:
/FederatedAccessManager-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 Federated Access Manager in the following directory:

/FederatedAccessManager-base`/SUNWam/web-src/services/css`.

The following table provides a brief description of each CSS file.

TABLE 16–4 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:

/FederatedAccessManager-base`/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 16–5 Sun Microsystems Branded GIF Images

File Name	Purpose
`Identity_LogIn.gif`	Sun Federated 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`	No longer used.

Localization Files

Localization files are located in the following directory:
/FederatedAccessManager-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 */FederatedAccessManager-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 */FederatedAccessManager-base*`/SUNWam/locale`.

TABLE 16–6 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 Federated Access Manager internal use only. Do not remove or modify this file.
`amAuthCert.properties`	Defines the Certificate Authentication Module.

TABLE 16–6 List of Localization Properties Files *(Continued)*

File Name	Purpose
`amAuthConfig.properties`	Defines the Authentication Configuration Module.
`amAuthContext.properties`	Defines the localized error messages for the `AuthContext` Java class.
`amAuthContextLocal.properties`	For Federated 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` */FederatedAccessManager-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:

/FederatedAccessManager-base`/SUNWam/web-src/services/config/auth/default`

to the new directory:

/FederatedAccessManager-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 265 and "Customizing JSP Templates" on page 265 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 277 in this chapter for instructions. See Chapter 14, "Updating and Redeploying Federated Access Manager WAR Files," for general information on updating and redeploying Federated Access Manager `.war` files.

6 **Restart both Federated 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 277 in this chapter for instructions. See Chapter 14, "Updating and Redeploying Federated Access Manager WAR Files," for general information on updating and redeploying Federated Access Manager `.war` files.

3 Restart both Federated 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 264 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 */FederatedAccessManager-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:

/FederatedAccessManager-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 Application Server documentation that comes with the IBM WebSphere product.

Customizing the Distributed Authentication User Interface

Federated 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 Federated 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 265.
- For information about customizing the `.xml` files, "XML Files" on page 267.

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.

◆ ◆ ◆ APPENDIX A

Key Management

A public key infrastructure enables users on a public network to securely and privately exchange data through the use of a public and a private key pair that is shared using a trusted authority. For example, the PKI allows the data from a client, such as a web browser, to be encrypted prior to transmission. The private key is used to decrypt text that has been encrypted with the public key. The public key is made publicly available (as part of a digital certificate) in a directory which all parties can access. This appendix contains information on how to create a keystore and generate public and private keys. It includes the following sections:

Public Key Infrastructure Basics

Web containers support the use of keystores to manage keys and certificates. The *keystore file* is a database that contains both public and private keys. Public and private keys are created simultaneously using the same algorithm (for example, RSA). A *public key* is used for encrypting or decrypting information. This key is made known to the world with no restrictions, but it cannot be used to decrypt information that the same key has encrypted. A *private key* is never revealed to anyone except it's owner and does not need to be communicated to third parties. The private key might never leave the machine or hardware token that originally generated it. The private key can encrypt information that can later be decrypted by using the public key. Also the private key can be used to decrypt information that was previously encrypted using the public key.

A public key infrastructure (PKI) is a framework for creating a secure method of exchanging information on an unsecure network. This ensures that the information being sent is not open to eavesdropping, tampering, or impersonation. It supports the distribution, management, expiration, rollover, backup, and revoking of the public and private keys used for public key cryptography. *Public key cryptography* is the most common method for encrypting and

decrypting a message. It secures the data involved in the communications by using a private key and its public counterpart. Each entity protects its own private key while disseminating its public key for all to use. Public and private keys operate inversely; an operation performed by one key can be reversed, or checked, only by its partner key.

Note – The Internet X.509 Public Key Infrastructure Certificate and CRL Profile is a PKI.

Digital Signatures

So, a private key and a public key can be used for simple message encryption and decryption. This ensures that the message can not be read (as in eavesdropping) but, it does not ensure that the message has not been tampered with. For this, a *one-way hash* (a number of fixed length that is unique for the data to be hashed) is used to generate a digital signature. A *digital signature* is basically data that has been encrypted using a one-way hash and the signer's private key. To validate the integrity of the data, the server receiving the communication uses the signer's public key to decrypt the hash. It then uses the same hashing algorithm that generated the original hash (sent with the digital signature) to generate a new one-way hash of the same data. Finally, the new hash and the received hash are compared. If the two hashes match, the data has not changed since it was signed and the recipient can be certain that the public key used to decrypt the digital signature corresponds to the private key used to create the digital signature. If they don't match, the data may have been tampered with since it was signed, or the signature may have been created with a private key that doesn't correspond to the public key presented by the signer. This interaction ensures that any change in the data, even deleting or altering a single character, results in a different value.

Digital Certificates

A *digital certificate* is an electronic document used to identify an individual, a server, a company, or other entity and to bind that entity to a public key by providing information regarding the entity, the validity of the certificate, and applications and services that can use the certificate. The process of signing the certificate involves tying the private key to the data being signed using a mathematical formula. The widely disseminated public counterpart can then be used to verify that the data is associated with the sender of the data. Digital certificates are issued by a certificate authority (CA) to authenticate the identity of the certificate-holder both before the certificate is issued and when the certificate is used. The CA can be either independent third parties or certificate-issuing server software specific to an enterprise. (Both types issue, verify, revoke and distribute digital certificates.) The methods used to authenticate an identity are dependant on the policies of the specific CA. In general, before issuing a certificate, the CA must use its published verification procedures for that type of certificate to ensure that an entity requesting a certificate is in fact who it claims to be.

Certificates help prevent the use of fake public keys for impersonation. Only the public key certified by the certificate will work with the corresponding private key possessed by the entity identified by the certificate. Digital certificates automate the process of distributing public keys and exchanging secure information. When one is installed on your machine, the public key is freely available. When another computer wants to exchange information with your computer, it accesses your digital certificate, which contains your public key, and uses it to validate your identity and to encrypt the information it wants to share with you. Only your private key can decrypt this information, so it remains secure from interception or tampering while traveling across the Internet.

Note – You can get a digital certificate by sending a request for one to a CA. Certificate requests are generated by the certificate management tool used. In this case, we are using the `keytool` command line interface. When `keytool` generates a certificate request, it also generates a private key.

`keytool` Command Line Interface

`keytool` is a key and certificate management utility used to create the keys. It also manages a `.keystore` file containing private keys and the associated X.509 certificate chains authenticating the corresponding public keys, issues certificate requests (which you send to the appropriate CA), imports certificate replies (obtained from the contacted CA), designates public keys belonging to other parties as trusted, and generates a unique key alias for each *keystore entry*. There are two types of entries in a keystore:

- A keystore entry holds sensitive cryptographic key information, stored in a protected format to prevent unauthorized access. Typically, a key stored in this type of entry is a secret or private key accompanied by a certificate chain for the corresponding public key.
- A trusted certificate entry contains a single public key certificate belonging to another party. It is called a *trusted certificate* because the keystore owner trusts that the public key in the certificate indeed belongs to the identity identified by the *subject* of the certificate. The issuer of the certificate vouches for this, by signing the certificate.

To create a keystore and default key entry in `.keystore`, you must use `keytool`, available from the Java Development Kit (JDK), version 1.3.1 and above. For more details, see keytool — Key and Certificate Management Tool.

Setting Up a Keystore

The following procedure illustrates how to create a keystore file and default key entry using `keytool`.

▼ To Set Up a Keystore

Be sure to use the `keytool` provided with the JDK bundled with Access Manager. It is located in *JAVA_HOME*`/bin/keytool`. When installed using the Java Enterprise System installer, *JAVA_HOME* is *AccessManager-base*`SUNWam/java`.

Note – The italicized option values in the commands used in this procedure may be changed to reflect your deployment.

---

1 Generate a certificate using one of the following procedures.

- **Generate a keystore with a public and private key pair and a self-signed certificate for your server using the following command.**

  ```
  keytool -genkey -keyalg rsa -alias test
  -dname "cn=sun-unix,ou=SUN Java System Access Manager,o=Sun,c=US"
  -keypass 11111111 -keystore .mykeystore
  -storepass 11111111 -validity 180
  ```

  This command will generate a keystore called `.mykeystore` in the directory from which it is run. A private key entry with the alias `test` is created and stored in `.mykeystore`. If you do not specify a path to the keystore, a file named `.keystore` will be generated in your home directory. If you do not specify an alias for the default key entry, `mykey` is created as the default alias. To generate a DSA key, change the value of `-keyalg` to `dsa`. This step generates a self-signed certificate.

- **Create a request and import a signed certificate from a CA (to authenticate your public key) using the following procedure.**

  a. **Create a request to retrieve a signed certificate from a CA (to authenticate your public key) using the following command:**

```
keytool -certreq -alias test -file request.csr -keypass 11111111 -keystore .mykeystore -storepass 11111111 -storetype JKS
```

  `.mykeystore` must also contain a self-signed certificate authenticating the server's generated public key. This step will generate the certificate request file, `request.csr`, under the directory from which the command is run. By submitting `request.csr` to a CA, the requestor will be authenticated and a signed certificate authenticating the public key will be returned. **[Remark A–1 Reviewer: Define the root certificate and the server**

certificate. How do you get both of these from one request?] Save this root certificate to a file named `myroot.cer` and save the server certificate generated in the previous step to a file named `mycert.cer`.

b. Import the certificate returned from the CA using the following command:

```
keytool -import -alias test -trustcacerts -file mycert.cer -keypass 11111111 -keystore .mykeystore -storepass 111111
```

c. Import the certificates of any trusted sites (from which you will receive assertions, requests and responses) into your keystore using the following command:

```
keytool -import -file myroot.cer -keypass 11111111 -keystore .mykeystore -storepass 11111111
```

The data to be imported must be provided either in binary encoding format, or in printable encoding format (also known as *Base64*) as defined by the Internet RFC 1421 standard. In the latter case, the encoding must be bounded at the beginning by a string that starts with `-----BEGIN` and bounded at the end by a string that starts with `-----END`.

2 **Change to the** *AccessManager-base*`/SUNWam/bin` **directory and run the following command:**

`ampassword -e` *original password*

[Remark A–2 Writer: Whose password is this encrypting?] This encrypts the password. The command will return something like `AQICKuNVNc9WXxiUyd8j9o/BR22szk8u69ME`.

3 **Create a new file named** `.storepass` **and put the encrypted password in it.**

4 **Create a new file named** `.keypass` **and put the encrypted password in it.**

5 **Copy** `.mykeystore` **to the location specified in** `AMConfig.properties`**.**

For example, if `com.sun.identity.saml.xmlsig.keystore=/etc/opt/SUNWam/lib/keystore.jks`, copy `.mykeystore` to `/etc/opt/SUNWam/lib/` and rename the file to `keystore.jks`.

6 **Copy** `.storepass` **and** `.keypass` **to the location specified in** `AMConfig.properties`**.**

For example, if `com.sun.identity.saml.xmlsig.storepass=/etc/opt/SUNWam/config/.storepass` and `com.sun.identity.saml.xmlsig.keypass=/etc/opt/SUNWam/config/.keypass`, copy both files to `/etc/opt/SUNWam/config/`.

7 **Define a value for the** `com.sun.identity.saml.xmlsig.certalias` **property in** `AMConfig.properties`**.**

For this example, the value would be `test`.

8 **(Optional) If the private key was encrypted using the DSA algorithm, change** `xmlsigalgorithm=http://www.w3.org/2000/09/xmldsig#rsa-sha1` **in**

AccessManager-base/locale/amSAML.properties **to** xmlsigalgorithm=http://www.w3.org/2000/09/xmldsig#dsa-sha1**.**

9 (Optional) Change the canonicalization method for signing or the transform algorithm for signing by modifying amSAML.properties**, located in** *AccessManager-base*/locale/**.**

a. Change canonicalizationMethod=http://www.w3.org/2001/10/xml-exc-c14n# **to any valid canonicalization method specified in Apache XML security package Version 1.0.5.**

Note – If this entry is deleted or left empty, we will use SAMLConstants.ALGO_ID_C14N_OMIT_COMMENTS (required by the XML Signature specification) will be used.

---

b. Change transformAlgorithm=http://www.w3.org/2001/10/xml-exc-c14n# **to any valid transform algorithm specified in Apache XML security package Version 1.0.5.**

Note – If this entry is deleted or left empty, the operation will not be performed.

---

10 Restart Access Manager.

Index

C

D

N

O

P

R

S

T

U

V

W

X