

Not for Publication

Sun Java System 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

Copyright 2007 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

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

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

This distribution may include materials developed by third parties.

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

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

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

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

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

Copyright 2007 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

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

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

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

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

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

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

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

List of Remarks

REMARK 5-1	Reviewer	Public or private? Doc or no?	10
REMARK 5-2	Reviewer	Public or private? Doc or no?	10
REMARK 5-3	Reviewer	Public or private? Doc or no?	10
REMARK 5-4	Reviewer	Public or private? Doc or no?	11
REMARK 5-5	Reviewer	Other packages used by WS-Federation??	12
REMARK 5-6	Reviewer	Other federation samples for WS-Federation??	13
REMARK 6-1	Reviewer	Is this still valid? Assuming you install client SDK for this.	16
REMARK 6-2	Reviewer	Is there still a saml2meta command line interface? Or has it been integrated into something else?	24
REMARK 6-3	Reviewer	Is there still a saml2meta command line interface? Or has it been integrated into something else?	24
REMARK 6-4	Reviewer	New	34
REMARK 6-5	Reviewer	New	35
REMARK 7-1	Reviewer	New	58
REMARK 7-2	Reviewer	New section.	65

Contents

5	Implementing Federation	9
	Understanding Federation	9
	Using the Federation API	10
	com.sun.identity.federation.accountmgmt	10
	com.sun.identity.federation.common	10
	com.sun.identity.federation.message	10
	com.sun.identity.federation.message.common	11
	com.sun.identity.federation.plugins	11
	com.sun.identity.federation.services	11
	com.sun.liberty	11
	Using the WS-Federation API	12
	com.sun.identity.ws federation.plugins	12
	com.sun.identity.ws federation.common	12
	Executing the Federation Samples	13
	sample1 Directory	13
	sample2 Directory	13
	sample3 Directory	14
6	Constructing SAML Messages	15
	SAML v2	15
	Using the SAML v2 SDK	15
	Service Provider Interfaces	18
	JavaServer Pages	24
	SAML 1.x	32
	Interfaces	32
	SAML 1.x Samples	38

7 Implementing Web Services	39
Developing New Web Services	39
▼ To Host a Custom Service	40
▼ To Invoke the Custom Service	47
Setting Up Liberty ID-WSF 1.1 Profiles	50
▼ To Configure Federated Access Manager to Use Liberty ID-WSF 1.1 Profiles	50
Common Application Programming Interfaces	56
Common Interfaces	56
Common Security API	58
Web Service Consumer Sample	59
Authentication Web Service	60
Authentication Web Service Default Implementation	60
Authentication Web Service API	61
Access the Authentication Web Service	62
Authentication Web Service Sample	62
Data Services	62
Liberty Personal Profile Service	63
Liberty Employee Profile Service	63
Data Services Template API	64
Discovery Service	65
Generating Security Tokens	65
▼ To Configure the Discovery Service to Generate Security Tokens	65
Discovery Service APIs	68
Access the Discovery Service	73
Discovery Service Sample	73
SOAP Binding Service	73
SOAPReceiver Servlet	73
SOAP Binding Service Package	74
Interaction Service	75
Configuring the Interaction Service	75
Interaction Service API	77
PAOS Binding	77
Comparison of PAOS and SOAP	78
PAOS Binding API	78
PAOS Binding Sample	79

Index83

Implementing Federation

Sun Java™ System Federated Access Manager has a robust federation framework for implementing federated identity infrastructures. It provides interfaces for creating, modifying, and deleting authentication domains, service providers, and identity providers and samples to get you started. This chapter covers the following topics:

- “Understanding Federation” on page 9
- “Using the Federation API” on page 10
- “Using the WS-Federation API” on page 12
- “Executing the Federation Samples” on page 13

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 Java System 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*.

Using the 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” on page 10
- “com.sun.identity.federation.common” on page 10
- “com.sun.identity.federation.message” on page 10
- “com.sun.identity.federation.message.common” on page 11
- “com.sun.identity.federation.plugins” on page 11
- “com.sun.identity.federation.services” on page 11
- “com.sun.liberty” on page 11

`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
<code>FSAttributeMapper</code>	Plug-in for mapping the attributes passed from the identity provider to local attributes on the service provider side during the single sign-on.
<code>FSAttributePlugin</code>	Plug-in for an identity provider to add <code>AttributeStatements</code> into a SAML assertion during the single sign-on process.
<code>FSIDPPProxy</code>	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
<code>getFederatedProviders()</code>	Returns a specific user's federated providers.
<code>getIDPFederationStatus()</code>	Retrieves a user's federation status with a specified identity provider. This method assumes that the user is already federated with the provider.
<code>getIDPList()</code>	Returns a list of all trusted identity providers.
<code>getIDPList()</code>	Returns a list of all trusted identity providers for the specified hosted provider.
<code>getProvidersToFederate()</code>	Returns a list of all trusted identity providers to which the specified user is not already federated.
<code>getSPList()</code>	Returns a list of all trusted service providers.
<code>getSPList()</code>	Returns a list of all trusted service providers for the specified hosted provider.
<code>getSPFederationStatus()</code>	Retrieves a user's federation status with a specified service provider. This method assumes that the user is already federated with the provider.

Using the WS-Federation API

[Remark 5-5 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.ws federation.plugins” on page 12
- “com.sun.identity.ws federation.common” on page 12

com.sun.identity.ws federation.plugins

Defines common WS-Federation utilities and constants.

com.sun.identity.ws federation.common

Defines WS-Federation Plugin SPIs

Executing the Federation Samples

[Remark 5–6 Reviewer: Other federation samples for WS-Federation??] The Federation samples are located in `/path-to-context-root/fam/samples/liberty`. To demonstrate the different Liberty-based federation protocols, three sample applications are included, and located in the following subdirectories:

- “sample1 Directory” on page 13
- “sample2 Directory” on page 13
- “sample3 Directory” on page 14

sample1 Directory

The sample1 directory provides a collection of files to configure a basic environment for creating and managing federation. The sample demonstrates the basic use of various Liberty-based federation protocols, including account federation, SSO, single logout, and federation termination. The scenario includes a service provider (SP), an identity provider (IDP), and configuration information for the two required servers. Each server must be deployed and configured on different installations of Federated Access Manager. Completing the procedures in the sample `Readme.txt` or `Readme.html` will help to give you a more complete understanding of how federation works.

TABLE 5–3 Configuration Information for sample1 Servers

Variable Placeholder	Host Name	Components Deployed on This Host
<i>machine1</i>	www.sp1.com	<ul style="list-style-type: none">▪ Service Provider▪ Web Service Consumer
<i>machine2</i>	www.idp1.com	<ul style="list-style-type: none">▪ Identity Provider▪ Discovery Service▪ Liberty Alliance Project

The `Readme.html` file in the sample1 directory provides detailed steps on how to deploy and configure this sample. sample1 also contains instructions for configuring a common domain. For information on common domains, see XXXXX.

sample2 Directory

The sample2 directory also provides a collection of files to configure a basic environment for creating and managing a federation. However, in this sample, the resources of the SP are deployed on an instance of Sun Java System Web Server that is protected by a policy agent. As in XXXXXsample1 Directory, the SP and IDP are deployed and configured on different

installations of Federated Access Manager. Besides demonstrating account federation, SSO, single logout, and federation termination, this sample also shows how different authentication contexts can be configured by associating different authentication levels with different protected pages. This association is made by creating policies for the protected resources. The `Readme.html` file in the `sample2` directory provides detailed steps on how to deploy and configure this sample.

sample3 Directory

The `sample3` directory provides a collection of files to configure an environment for creating and managing a federation that includes two SPs and two IDPs. In this sample, all providers are hosted on a single installation of Federated Access Manager. You need to host the same IP address (the one on which Federated Access Manager is installed) in four different DNS domains. Thus, four virtual server instances are created on separate instances of Sun Java System Web Server, one for each of the providers.

Note – Virtual server instances can be simulated by adding entries in the `/etc/hosts` file for the fully qualified host names of the virtual servers.

Because this scenario involves multiple IPs, you also need to install a common domain. You can install the Common Domain Services for Federation Management on the same machine as the Federated Access Manager software or on a different machine. The `Readme.html` file in the `sample3` directory provides detailed steps on how to deploy and configure this sample. You can also find information about common domains in XXXXX.

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” on page 15](#)
- [“SAML 1.x” on page 32](#)

SAML v2

- [“Using the SAML v2 SDK” on page 15](#)
- [“Service Provider Interfaces” on page 18](#)
- [“JavaServer Pages” on page 24](#)

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 15](#).
- For ways to set a customized implementation, see [“Setting a Customized Class” on page 16](#).
- For instructions on how to install the SDK as a standalone application, see [“Installing the SAML v2 SDK” on page 16](#).

Exploring the SAML v2 Packages

The SAML v2 SDK includes the following packages:

- [“com.sun.identity.saml2.assertion Package” on page 16](#)

- “`com.sun.identity.saml2.common` Package” on page 16
- “`com.sun.identity.saml2.protocol` Package” on page 16

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

The following procedure contains instructions to install the SAML v2 SDK.

Remark 6–1 Reviewer

Is this still valid? Assuming you install client SDK for this.

▼ To Install the SAML v2 SDK

Before You Begin If installing the SDK on a Linux system, you must have the Red Hat Package Manager (RPM) installed.

1 Log in as root.

2 Create a new directory.

```
# mkdir saml2bits
```

```
# cd saml2bits
```

3 Download the *file-name.tar.gz* file into the new directory.

4 Unpack the product binaries by typing:

```
# gunzip -dc file-name.tar.gz | tar -xvof -
```

where *file-name.tar.gz* is the name of the downloaded file.

5 Add the SAML v2 packages as follows:

```
# pkgadd -d . SUNWsaml2
```

By default, the packages will be installed in */path-to-context-root/fam/saml2*.

6 Add the following to the classpath of your application:

- **For Access Manager 7 2005Q4:**

- */opt/product-directory/saml2/lib/saml2.jar*
- */opt/product-directory/saml2/locale*

- **For Federation Manager 7 2005Q4:**

- */opt/SUNWam/saml2/lib/saml2.jar*
- */opt/SUNWam/saml2/locale*

7 Get the supporting JAR and locale files using the applicable procedure:

- **For Access Manager 7 2005Q4:**

- a. `cd /AccessManager-base/product-directory`

- b. **Run the following command:**

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

- c. **Add the following to the classpath of your application:**

- `AccessManager-base/product-directory/clientsdk-webapps/WEB-INF/lib/amclientsdk.jar`
 - `AccessManager-base/product-directory/clientsdk-webapps/WEB-INF/classes`
- **For Federation Manager 7 2005Q4:**
Add the following to the classpath of your application:
 - a. `FederationManager-base/SUNWam/fm/web-src/WEB-INF/lib/am_services.jar`
 - b. `FederationManager-base/SUNWam/fm/web-src/WEB-INF/lib/am_sdk.jar`
 - c. `FederationManager-base/SUNWam/fm/web-src/WEB-INF/classes`

8 Restart your application.

You should now be able to process SAML v2 XML messages using the methods in the `AssertionFactory` and `ProtocolFactory`.

Next Steps For details regarding the SAML v2 SDK classes, see the *Federated Access Manager 8.0 Java API Reference*.

Service Provider Interfaces

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 18](#)
- [“Attribute Mappers” on page 19](#)
- [“Authentication Context Mappers” on page 20](#)

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 19](#)
- [“SPAccountMapper” on page 19](#)

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 `<AttributeStatement>`.

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 19](#)
- [“SPAttributeMapper” on page 19](#)
- [“To Set Up Attribute Mappers” on page 20](#)

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.

▼ To Set Up Attribute Mappers

This procedure will pass the `mail` and `employeeNumber` attributes from the identity provider to the service provider.

- 1 **Export the identity provider's current extended metadata configuration to a file.**

```
saml2meta [-i staging-directory] export -u amadmin -w password -e IDP-entityID -x  
IDP-extended-XML-file-name
```

- 2 **Edit the `attributeMap` attribute in the exported extended metadata configuration file to include the user attributes the identity provider will pass to the service provider.**

`attributeMap` defines the mapping between the provider that this metadata is configuring and the remote provider. This attribute takes a value of *autoFedAttribute-value=remote-provider-attribute*. For example,

```
<Attribute name="attributeMap">  
<Value>mail=mail</Value>  
<Value>employeeNumber=employeeNumber</Value>  
</Attribute>
```

- 3 **Remove the identity provider's current extended metadata configuration.**

```
saml2meta [-i staging-directory] delete -u amadmin -w password -e IDP-entityID -c
```

- 4 **Import the identity provider's modified extended metadata configuration file.**

```
saml2meta [-i staging-directory] import -u amadmin -w password -x  
IDP-extended-XML-file-name
```

- 5 **Restart the web container.**

- 6 **Repeat the above steps for the service provider's extended metadata configuration file.**

- 7 **To test, invoke single sign-on from the service provider.**

The assertion contains an `AttributeStatement` with the `mail` and `employeeNumber` attributes which will be set in the single sign-on token.

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=PasswordProtectedTransport`

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 22](#)
- [“SPAuthnContextMapper” on page 22](#)

The procedure for configuring mappings is in [“To Configure Mappings” on page 23](#).

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 | authnType=authnValue | authnType=authnValue | ...`

For example,

`urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport|module=LDAP`
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*.

▼ To Configure Mappings

The following procedure assumes you are mapping

`urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport` to authentication level 4 on the service provider and use the LDAP authentication module for authentication on the identity provider.

- 1 **Set the mapping for the `spAuthnContextClassRefMapping` property in the current extended service provider metadata.**

For example, `PasswordProtectedTransport|4`

- 2 Reload the modified metadata using `saml2meta`.

[Remark 6–2 Reviewer: Is there still a `saml2meta` command line interface? Or has it been integrated into something else?] See XXXXXThe `saml2meta` Command-line Reference.

- 3 Set the mapping for the `idpAuthnContextClassRefMapping` property in the current extended identity provider metadata.

For example,

```
urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport|module=LDAP
```

- 4 Reload the modified metadata using `saml2meta`.

[Remark 6–3 Reviewer: Is there still a `saml2meta` command line interface? Or has it been integrated into something else?] See XXXXXThe `saml2meta` Command-line Reference.

- 5 Access the single sign-on initialization page using the following URL:

`http://AM_host:AM_port/uri/spSSOinit.jsp?`

`metaAlias=/sp&idpEntityID=idp.sun.com&AuthnContextClassRef=PasswordProtectedTransport`

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.

- “Default Display Page” on page 25
- “Assertion Consumer Page” on page 25
- “Single Sign-on Pages” on page 25
- “Name Identifier Pages” on page 28
- “Single Logout JavaServer Pages” on page 30



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 26](#)
- [“idpSSOInit.jsp” on page 26](#)

- “spSSOInit.jsp” on page 26

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 than 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/spsssoinit*. 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. 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 create 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 28](#)
- [“idpMNIRedirect.jsp” on page 29](#)
- [“spMNIRequestInit.jsp” on page 29](#)
- [“spMNIRedirect.jsp” on page 29](#)

`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 30](#)
- [“idpSingleLogoutRedirect.jsp” on page 31](#)
- [“spSingleLogoutInit.jsp” on page 31](#)
- [“spSingleLogoutRedirect.jsp” on page 31](#)

`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 1.x

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

- [“Interfaces” on page 32](#)
- [“SAML 1.x Samples” on page 38](#)

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:

- [“`com.sun.identity.saml` Package” on page 32](#)
- [“`com.sun.identity.saml.assertion` Package” on page 33](#)
- [“`com.sun.identity.saml.common` Package” on page 33](#)
- [“`com.sun.identity.saml.plugins` Package” on page 33](#)
- [“`com.sun.identity.saml.protocol` Package” on page 35](#)
- [“`com.sun.identity.saml.xmlsig` Package” on page 38](#)

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” on page 32](#)
- [“`SAMLClient` Class” on page 33](#)

`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 6-1 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 34
- “[AttributeMapper Interface](#)” on page 34
- “[NameIdentifierMapper Interface](#)” on page 34
- “[PartnerAccountMapper Interface](#)” on page 34
- “[PartnerSiteAttributeMapper Interface](#)” on page 35

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 6–4 Reviewer



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 6–5 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 XXXXXX 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](#)” on page 35
- “[AttributeQuery Class](#)” on page 36
- “[AuthorizationDecisionQuery Class](#)” on page 36

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 6-2 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=admin,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);
```

EXAMPLE 6-2 AuthorizationDecisionQuery Code Sample (Continued)

```
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 38](#).

SAML 1.x Samples

You can access several SAML-based samples from the [“SAML 1.x Samples” on page 38](#) 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.

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” on page 39
- “Setting Up Liberty ID-WSF 1.1 Profiles” on page 50
- “Common Application Programming Interfaces” on page 56
- “Web Service Consumer Sample” on page 59
- “Authentication Web Service” on page 60
- “Data Services” on page 62
- “Discovery Service” on page 65
- “SOAP Binding Service” on page 73
- “Interaction Service” on page 75
- “PAOS Binding” on page 77

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 40](#)
- [“To Invoke the Custom Service” on page 47](#)

▼ 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" minOccurs="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.

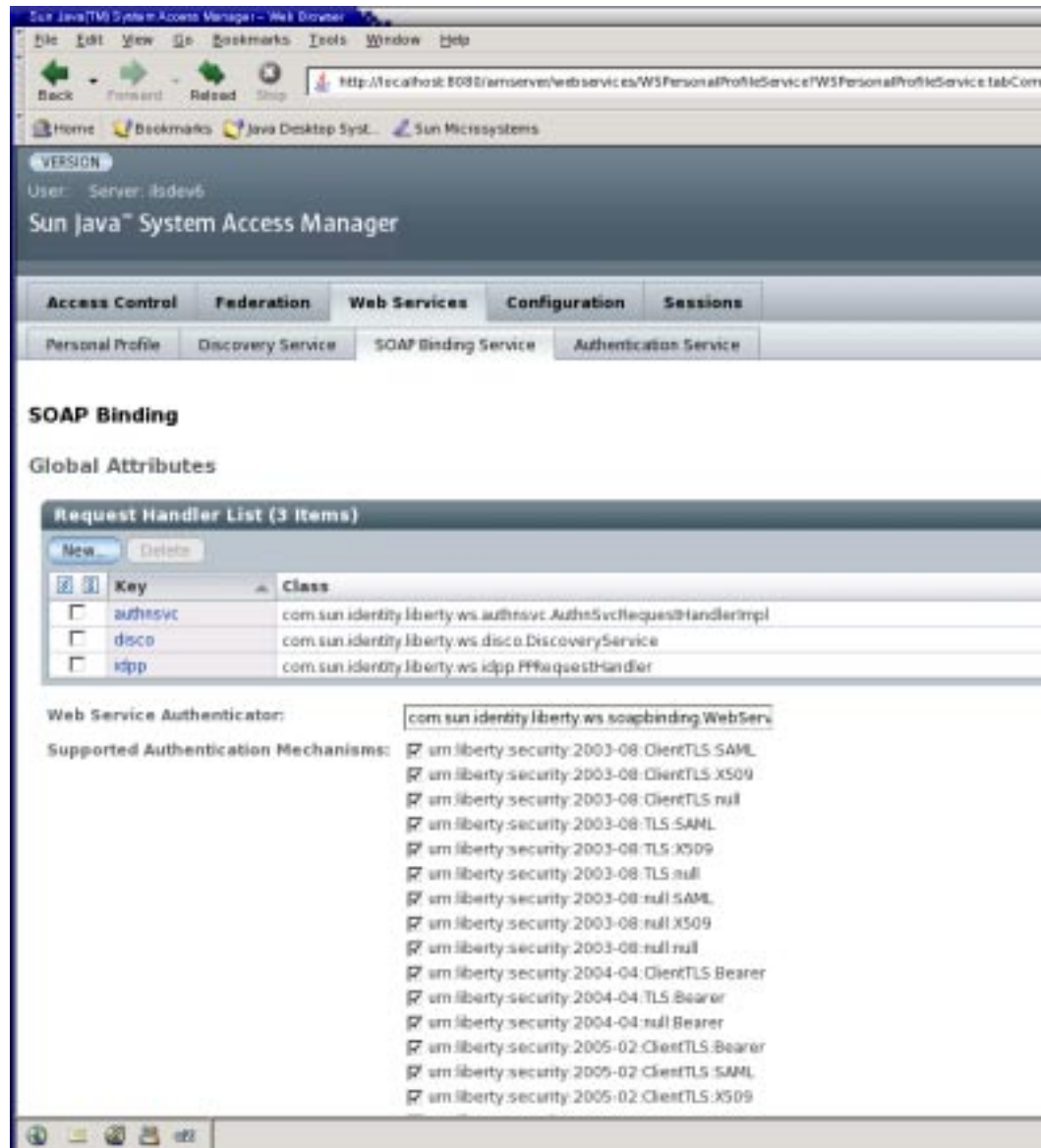
4 Add the previously created classes to the web container classpath and restart the web container.

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.

The screenshot shows a web browser window titled "Sun Java™ System Access Manager - Web Browser". The address bar shows the URL "http://localhost:8080/iamserver/realms/RealmResourceOffering". The browser's toolbar includes buttons for Back, Forward, Reload, and Stop. Below the browser window, the page header displays "User: Server: jsdev6" and the title "Sun Java™ System Access Manager".

The main content area is titled "New Resource Offerings". It contains a "Description:" text input field. Below this is the "Service Instance" section, which includes:

- * Service Type: [Text Input Field] (URI defining the type of service this service instance implements.)
- * Provider ID: [Text Input Field] (URI of the provider of the service instance.)
- * It is required to define 1 or more service descriptions.

Below the service instance section is the "Service Description (0 Items)" section. It contains a "New Description..." button and a "Delete" button. Below these buttons is the "Security Mechanism ID" section, which states "There are no descriptions defined".

The "Resource Offering Options" section includes:

- Options: ☐ Service has no options to advertise.
- Option List: A list box labeled "Current Values" with a "Remove" button next to it.
- New Value: A text input field with an "Add" button next to it.

The bottom of the page shows a "Disclaimer" section with a "Done" button.

- 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(disco.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);

```

```
// Uncomment 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.ipplanet.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.

- “Common Interfaces” on page 56
- “Common Security API” on page 58

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” on page 56
- “com.sun.identity.liberty.ws.interfaces Package” on page 56

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

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

TABLE 7-1 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 7-2 com.sun.identity.liberty.ws.interfaces Interfaces

Interface	Description
Authorizer	<p>This interface, once implemented, can be used by each Liberty-based web service component for access control.</p> <p>Note – The <code>com.sun.identity.liberty.ws.disco.plugins.DefaultDiscoAuthorizer</code> class is the implementation of this interface for the Discovery Service. For more information, see XXXXX. The <code>com.sun.identity.liberty.ws.idpp.plugin.IDPPAuthorizer</code> class is the implementation for the Liberty Personal Profile Service. For more information, see XXXXX.</p> <p>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 <i>invocation identity</i> 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.</p> <p>Note – See the <i>Sun Java System Federated Access Manager 8.0 Technical Overview</i> for more information about policy management, single sign-on, and user sessions. See the XXXXX for information about creating policy.</p>
ResourceIDMapper	<p>This interface is used to map a user DN to the resource identifier associated with it. Federated Access Manager provides implementations of this interface.</p> <ul style="list-style-type: none"> ■ <code>com.sun.identity.liberty.ws.disco.plugins.Default64ResourceIDMapper</code> assumes the Resource ID format to be: <i>providerID + "/" + the Base64 encoded userIDs</i>. ■ <code>com.sun.identity.liberty.ws.disco.plugins.DefaultHexResourceIDMapper</code> assumes the Resource ID format to be: <i>providerID + "/" + the hex string of userID</i>. ■ <code>com.sun.identity.liberty.ws.idpp.plugin.IDPPResourceIDMapper</code> assumes the Resource ID format to be: <i>providerID + "/" + the Base64 encoded userIDs</i>. <p>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 <i>providerID</i> and the implementation class can be configured through the Classes For ResourceIDMapper Plugin attribute.</p>
ServiceInstanceUpdate	<p>Interface used to include a SOAP header (<code>ServiceInstanceUpdateHeader</code>) when sending a SOAP response.</p>

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 7-1
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 7-3 `com.sun.identity.liberty.ws.security` Classes

Class	Description
<code>ProxySubject</code>	Represents the identity of a proxy, the confirmation key, and confirmation obligation the proxy must possess and demonstrate for authentication purposes.
<code>ResourceAccessStatement</code>	Conveys information regarding the accessing entities and the resource for which access is being attempted.
<code>SecurityAssertion</code>	Provides an extension to the <code>Assertion</code> class to support ID-WSF <code>ResourceAccessStatement</code> and <code>SessionContextStatement</code> .
<code>SecurityTokenManager</code>	An entry class for the security package <code>com.sun.identity.liberty.ws.security</code> . 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.
<code>SecurityUtils</code>	Defines methods that are used to get certificates and sign messages.
<code>SessionContext</code>	Represents the session status of an entity to another system entity.
<code>SessionContextStatement</code>	Conveys the session status of an entity to another system entity within the body of an <code><saml:assertion></code> element.
<code>SessionSubject</code>	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 7-4 `com.sun.identity.liberty.ws.common.wsse` Classes

Class	Description
<code>BinarySecurityToken</code>	Provides an interface to parse and create the X.509 Security Token depicted by Web Service Security: X.509
<code>WSSEConstants</code>	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” on page 60
- “Authentication Web Service API” on page 61
- “Access the Authentication Web Service” on page 62
- “Authentication Web Service Sample” on page 62

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 7-5 Default Implementations for Authentication Mechanism

Class	Description
<code>com.sun.identity.liberty.ws.authnsvc.mechanism.PlainMechanismHandler</code>	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.
<code>com.sun.identity.liberty.ws.authnsvc.mechanism.CramMD5MechanismHandler</code>	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 `com.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 61
- “[com.sun.identity.liberty.ws.authnsvc.mechanism Package](#)” on page 61
- “[com.sun.identity.liberty.ws.authnsvc.protocol Package](#)” on page 62

`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” on page 63
- “Liberty Employee Profile Service” on page 63
- “Data Services Template API” on page 64

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 64
- “com.sun.identity.liberty.ws.dst.service Package” on page 64

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 7-6 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 7–2 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 `XXXXXXEntities` 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 `XXXXXXAuthentication 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 69.](#)
- `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 70.](#)
- `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 70](#) and [“`com.sun.identity.liberty.ws.interfaces.ResourceIDMapper` Interface” on page 72.](#)

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 XXXXXXChapter 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 7-7 Discovery Service Client APIs

Class	Description
Description	Represents a <code>DescriptionType</code> element of a service instance.
Directive	Represents a discovery service <code>DirectiveType</code> element.
DiscoveryClient	Provides methods to send Discovery Service queries and modifications.
EncryptedResourceID	Represents an <code>EncryptionResourceID</code> 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 complies 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 7-8 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 <code>sunIdentityServerDiscoEntries</code> attribute. The <code>UserDiscoEntryHandler</code> 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 <code>sunIdentityServerDynamicDiscoEntries</code> dynamic attribute in the Discovery Service. Modification of these entries is not supported and always returns <code>false</code> . The resource offering is saved in an organization or a role. The <code>DynamicDiscoEntryHandler</code> 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 <code>sunIdentityServerDiscoEntries</code> attribute and discovery entries stored in the Discovery Service <code>sunIdentityServerDynamicDiscoEntries</code> attribute. It modifies only discovery entries stored in the user entry. The <code>UserDynamicDiscoEntryHandler</code> 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 XXXXXClassesForResourceIDMapper 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 73](#)
- [“SOAP Binding Service Package” on page 74](#)

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 7-9 SOAP Binding Service API

Class	Description
<code>Client</code>	Provides a method with which a WSC can send a request to a WSP using a SOAP connection. It also returns the response.
<code>ConsentHeader</code>	Represents the SOAP element named <code>Consent</code> .
<code>CorrelationHeader</code>	Represents the SOAP element named <code>Correlation</code> . By default, <code>CorrelationHeader</code> will always be signed.
<code>ProcessingContextHeader</code>	Represents the SOAP element named <code>ProcessingContext</code> .
<code>ProviderHeader</code>	Represents the SOAP element named <code>Provider</code> .
<code>RequestHandler</code>	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.
<code>Message</code>	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.
<code>ServiceInstanceUpdateHeader</code>	Allows a service to change the endpoint on which requesters will contact it.
<code>ServiceInstanceUpdateHeader.Credential</code>	Allows a service to use a different security mechanism and credentials to access the requested resource.
<code>SOAPFault</code>	Represents the SOAP element named <code>SOAP Fault</code> .
<code>SOAPFaultDetail</code>	Represents the SOAP element named <code>Detail</code> , a child element of <code>SOAP Fault</code> .
<code>UsageDirectiveHeader</code>	Defines the SOAP element named <code>UsageDirective</code> .

See “[PAOS Binding](#)” on page 77 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” on page 75
- “Interaction Service API” on page 77

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 7-10 Interaction Service Properties in `AMConfig.properties`

Property	Description
<code>com.sun.identity.liberty.interaction.wspRedirectHandler</code>	Points to the URL where the <code>WSPRedirectHandler</code> servlet is deployed. The servlet handles the service provider side of interactions for user redirects.
<code>com.sun.identity.liberty.interaction.wscSpecifiedInteractionChoice</code>	Indicates the level of interaction in which the WSC will participate if the WSC participates in user redirects. Possible values include <code>interactIfNeeded</code> , <code>doNotInteract</code> , and <code>doNotInteractForData</code> . The affirmative <code>interactIfNeeded</code> is the default.
<code>com.sun.identity.liberty.interaction.wscWillIncludeUserInteractionHeader</code>	Indicates whether the WSC will include a SOAP header to indicate certain preferences for interaction based on the Liberty specifications. The default value is <code>yes</code> .
<code>com.sun.identity.liberty.interaction.wscWillRedirect</code>	Indicates whether the WSC will participate in user redirections. The default value is <code>yes</code> .

TABLE 7-10 Interaction Service Properties in AMConfig.properties (Continued)

Property	Description
<code>com.sun.identity.liberty.interaction.wscSpecifiedMaxInteractionTime</code>	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 <code>com.sun.identity.liberty.interaction.wspRedirectTime</code> is set to 40 seconds, the WSP returns a SOAP fault (<code>timeNotSufficient</code>), indicating that the time is insufficient for interaction.
<code>com.sun.identity.liberty.interaction.wspWillEnforceHttpsCheck</code>	Indicates whether the WSC will enforce HTTPS in redirected URLs. The Liberty Alliance Project specifications state that, the value of this property is always <code>yes</code> , which indicates that the WSP will not redirect the user when the value of <code>redirectURL</code> (specified by the WSP) is not an HTTPS URL. The <code>false</code> value is primarily meant for ease of deployment in a phased manner.
<code>com.sun.identity.liberty.interaction.wspWillRedirect</code>	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 <code>yes</code> .
<code>com.sun.identity.liberty.interaction.wspWillRedirectForData</code>	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 <code>yes</code> .
<code>com.sun.identity.liberty.interaction.wspRedirectTime</code>	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 <code>com.sun.identity.liberty.interaction.wscSpecifiedMaxInteractionTime</code>) for interaction. If the <code>wspRedirectTime</code> is set to 40 seconds, the WSP returns a SOAP fault (<code>timeNotSufficient</code>), indicating that the time is insufficient for interaction.
<code>com.sun.identity.liberty.interaction.wspWillEnforceHttpsCheck</code>	Indicates whether the WSP will enforce a HTTPS <code>returnToURL</code> specified by the WSC. The Liberty Alliance Project specifications state that the value of this property is always <code>yes</code> . The <code>false</code> value is primarily meant for ease of deployment in a phased manner.

TABLE 7-10 Interaction Service Properties in AMConfig.properties (Continued)

Property	Description
com.sun.identity.liberty. interaction. wspWillEnforceReturnToHost EqualsRequestHost	Indicates whether the WSP would enforce the address values of <code>returnToHost</code> and <code>requestHost</code> 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 7-11 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” on page 78](#)
- [“PAOS Binding API” on page 78](#)
- [“PAOS Binding Sample” on page 79](#)

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 7–12 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 7-12 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 7-1 PAOS Client Servlet From PAOS Sample

```
import java.util.*;
import java.io.*;

import javax.servlet.*;
import javax.servlet.http.*;
```

EXAMPLE 7-1 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 7-1 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 (PAOException 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 (PAOException 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();
    }
}

```


Index

A

access

- Authentication Web Service, 62
- Discovery Service, 73

account mappers, 18-19

API

- Authentication Web Service, 60-62
- client for Discovery Service, 69-70
- common security, 58-59
- common service, 56-58
- Data Services Template, 64-65
- Discovery Service, 68-73
- federation, 10-12
- Interaction Service, 75-77
- PAOS binding, 77-81
- SAML 1.x, 32-38
- SOAP Binding Service, 74
- WS-Federation, 12

attribute mappers, 19-20

- setting up, 20

attributes, Authentication Web Service, 60-61

authentication context mappers, 20-24

Authentication Web Service

- accessing, 62
- API, 60-62
- attribute, 60-61
- sample, 62
- XML service file, 60-61

Authorizer interface, 70-72

Authorizer interface, 57

C

client API

- Data Services Template, 64
- Discovery Service, 69-70

com.sun.identity.federation.plugins, 11

com.sun.identity.federation.services, 11

com.sun.identity.liberty.wsf.version, 50-56

com.sun.identity.saml2.assertion, 16

com.sun.identity.saml2.common, 16

com.sun.identity.saml2.protocol, 16

com.sun.liberty, 11-12

common interfaces, 56-59

common security API, 58-59

D

data services

- API, 64-65
- Liberty Employee Profile Service, 63
- Liberty Personal Profile Service, 63

Data Services Template

- API, 64-65
- client API, 64

default.jsp, 25

Default64ResourceIDMapper, 72-73

DefaultDiscoAuthorizer class, 70-72

DefaultHexResourceIDMapper, 72-73

develop web services, invoke, 47-49

DiscoEntryHandler interface, 70

Discovery Service

- accessing, 73

Discovery Service (*Continued*)

- and policy creation, 70-72
- and security tokens, 65-68
- API, 68-73
- client API, 69-70
- sample, 73

E

- employee profile service sample, 63

F**federation**

- API, 10-12
- common interfaces, 56-59
- samples, 13-14

I

- idpMNIRedirectInit.jsp, 29
- idpMNIRestInit.jsp, 28
- idpSingleLogoutInit.jsp, 30
- idpSingleLogoutRedirect.jsp, 31
- idpSSOFederate.jsp, 26
- idpSSOInit.jsp, 26
- installation, SAML v2 SDK, 17-18
- Interaction Service, 75-77
- interfaces
 - Authentication Web Service, 60-62
 - Authorizer, 70-72
 - DiscoEntryHandler, 70
 - Discovery Service, 68-73
 - request handler, 74
 - ResourceIDMapper, 72-73

J

- JSP, SAML v2, 24-32

L

- Liberty Employee Profile Service, 63
- Liberty ID-WSF 1.1 profiles, 50-56
- Liberty Personal Profile Service, 63

O**overview**

- Liberty Employee Profile Service, 63
- Liberty Personal Profile Service, 63

P

- PAOS binding, 77-81
 - PAOS or SOAP, 78
 - sample, 79-81
- policy creation, and Discovery Service, 70-72
- procedures, create policy for
 - DefaultDiscoAuthorizer, 70-72
- profiles, set up Liberty ID-WSF, 50-56

R

- RelayState, 25
- RequestHandler interface, 65
- ResourceIDMapper interface, 72-73
- ResourceIDMapper interface, 57

S

- SAML, samples, 38
- SAML 1.x, API, 32-38
- SAML v2
 - adding implementation class, 15-18
 - com.sun.identity.saml2.assertion, 16
 - com.sun.identity.saml2.common, 16
 - com.sun.identity.saml2.protocol, 16
 - default.jsp, 25
 - idpMNIRedirectInit.jsp, 29
 - idpMNIRestInit.jsp, 28
 - idpSingleLogoutInit.jsp, 30

SAML v2 (Continued)

- idpSingleLogoutRedirect.jsp, 31
- idpSSOFederate.jsp, 26
- idpSSOInit.jsp, 26
- JavaServer Pages, 24-32
- SDK, 15-18
- SDK installation, 17-18
- spAssertionConsumer.jsp, 25
- SPI, 18-24
- spMNIRedirect.jsp, 29-30
- spMNIRedirectInit.jsp, 29
- spSingleLogoutInit.jsp, 31
- spSingleLogoutRedirect.jsp, 31-32
- spSSOInit.jsp, 26-28

samples

- Authentication Web Service, 62
- Discovery Service, 73
- employee profile service, 63
- federation, 13-14
- PAOS binding, 79-81
- SAML, 38
- security tokens, 65-68
- web service consumer, 59

SDK, SAML v2, 15-18

security tokens

- and Discovery Service, 65-68
- generating, 65-68

SOAP Binding Service

- API, 74
- PAOS or SOAP, 78
- SOAPReceiver, 73

SOAPReceiver, 73

spAssertionConsumer.jsp, 25

SPI

- account mappers, 18-19
- attribute mappers, 19-20
- authentication context mappers, 20-24
- SAML v2, 18-24

spMNIRedirect.jsp, 29-30

spMNIRedirectInit.jsp, 29

spSingleLogoutInit.jsp, 31

spSingleLogoutRedirect.jsp, 31-32

spSSOInit.jsp, 26-28

U

use cases, set up attribute mappers, 20

W

web service consumer sample, 59

web services

- develop, 39-49
- hosting, 40-47
- invoking, 47-49

WS-Federation, API, 12

X

XML service files, Authentication Web Service, 60-61

