Sun Java System Access Manager 7 2005Q4 Federation and SAML Administration Guide

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

Part No: 819–2142–11
October 2005

060815@15490

Contents

Figures

Tables

Examples

Preface

The *Sun™ Java System Access Manager 7 2005Q4 Federation and SAML Administration Guide* provides information about the Federation and Security Assertions Markup Language (SAML) components of Sun™ Java System Access Manager. The *Federation and SAML Administration Guide* includes an introduction to the open-standard specifications used to develop these features and information on how Access Manager has implemented them. It also includes information on integrated web services, and summaries of the application programming interface (API).

Who Should Use This Book

This *Federation and SAML Administration Guide* is intended for use by IT professionals, network administrators and software developers who implement a Liberty-enabled identity framework and access platform using Sun Java System servers and software. It is recommended that administrators understand the following technologies:

- Lightweight Directory Access Protocol (LDAP)
- Java™
- JavaServer Pages™ (JSP)
- HyperText Transfer Protocol (HTTP)
- HyperText Markup Language (HTML)
- eXtensible Markup Language (XML)
- Web Services Description Language (WSDL)
- Security Assertion Markup Language (SAML)
- SOAP (SOAP is no longer an acronym for the messaging protocol.)

Before You Read This Book

Access Manager is a component of the Sun Java Enterprise System, a software infrastructure that supports enterprise applications distributed across a network or Internet environment.

- Because Access Manager is a component of the Sun Java Enterprise System, you should be familiar with the Sun Java Enterprise System 2005Q4 documentation set.
- Because Sun Java System Directory Server is used as the data store in a new Access Manager deployment, you should be familiar with the Sun Java System Directory Server 5 2005Q4 documentation set..
- Because Access Manager contains features based on the Liberty Alliance Project specifications, you should be familiar with the Liberty Alliance Project specifications.

How This Book Is Organized

The *Federation and SAML Administration Guide* contains instructional and conceptual material regarding the Access Manager features based on the Liberty Alliance Project and SAML specifications. The book is organized into the chapters described in the following table.

TABLE P–1 Chapters in Federation and SAML Administration Guide

Chapter	Description
Chapter 1, Introduction to the Liberty Alliance Project	An overview of the specifications developed by the Liberty Alliance Project.
Chapter 2, Implementation of the Liberty Alliance Project Specifications	Contains conceptual material regarding the implementation of the Liberty Alliance Project specifications in Access Manager and its architecture.
Chapter 3, Federation	Provides administrative information regarding setting up entities and authentication domains as well as information on extended federation capabilities.
Chapter 4, Common Domain Services	Provides information regarding the installation and deployment of the Common Domain Services.
Chapter 5, Authentication Web Service	Provides information regarding the deployment of the Authentication Web Service.
Chapter 6, Data Services	Provides information regarding data services in general and the Liberty Personal Profile Service and Liberty Employee Profile Service in particular.
Chapter 7, Discovery Service	Provides information regarding the administration and deployment of the Discovery Service.
Chapter 8, SOAP Binding Service	Provides information regarding the administration and deployment of the SOAP Binding Service.
Chapter 9, SAML Administration	Provides information regarding the implementation of SAML in Access Manager functions.
Chapter 10, Application Programming Interfaces	Provides information regarding the API developed for Access Manager that are based on the Liberty Alliance Project specifications.
Appendix A, Access Manager Samples	An appendix that provides information on the samples developed for Access Manager and based on the Liberty Alliance Project specifications.

TABLE P–1 Chapters in Federation and SAML Administration Guide *(Continued)*

Chapter	Description
Appendix B, Service Schema Files	An appendix that contains the XML Schema Definition (XSD) files developed by the Liberty Alliance Project. The XSD files specify the information its corresponding service can host by defining the data and data structure.

Related Books

The Access Manager documentation consists of two sets:

- "Access Manager Core Documentation" on page 19
- "Sun Java System Product Documentation" on page 20

Note – For instructions on installing Access Manager, see the *Sun Java Enterprise System 2005Q4 Installation Guide for UNIX*.

Access Manager Core Documentation

The Access Manager documentation set contains the following titles:

- The *Sun Java System Access Manager 7 2005Q4 Release Notes* will be available online after the product is released. It gathers an assortment of last-minute information, including a description of what is new in this current release, known problems and limitations, installation notes, and how to report issues with the software or the documentation.
- The *Sun Java System Access Manager 7 2005Q4 Technical Overview* provides an overview of how Access Manager components work together to consolidate access control functions, and to protect enterprise assets and web-based applications. It also explains basic Access Manager concepts and terminology.
- The *Sun Java System Access Manager 7 2005Q4 Deployment Planning Guide* provides information for planning an Access Manager deployment within an existing information technology infrastructure.
- The *Sun Java System Access Manager 7 2005Q4 Performance Tuning Guide* provides information on how to tune Access Manager and its related components for optimal performance.
- The *Sun Java System Access Manager 7 2005Q4 Administration Guide* describes how to use the Access Manager console as well as manage user and service data via the command line interface.
- The *Sun Java System Access Manager 7 2005Q4 Federation and SAML Administration Guide* (this guide) provides information about the features in Access Manager that are based on the Liberty Alliance Project and SAML specifications. It includes information on the integrated services based on these specifications, instructions for enabling a Liberty-based environment, and summaries of the application programming interface (API) for extending the framework.

- The *Sun Java System Access Manager 7 2005Q4 Developer's Guide* offers information on how to customize Access Manager and integrate its functionality into an organization's current technical infrastructure. It also contains details about the programmatic aspects of the product and its API.
- The *Sun Java System Access Manager 7 2005Q4 C API Reference* provides summaries of data types, structures, and functions that make up the public Access Manager C APIs.
- The *Java API Reference* are generated from Java code using the Javadoc™ tool. The pages provide information on the implementation of the Java packages in Access Manager.
- The *Sun Java System Access Manager Policy Agent 2.2 User's Guide* provides an overview of the policy functionality and the policy agents available for Access Manager.

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

Sun Java System Product Documentation

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

- Sun Java System Directory Server
- Sun Java System Web Server
- Sun Java System Application Server
- Sun Java System Web Proxy Server

Accessing Sun Resources Online

For product downloads, professional services, patches, support, and additional developer information, go to:

- Download Center
- Sun Software Services
- Sun Java Systems Services Suite
- Sun Enterprise Services, Solaris Patches, and Support
- Developer Information

Contacting Sun Technical Support

If you have technical questions about this product that are not answered in the product documentation, contact Sun Support Services.

Related Third-Party Web Site References

Third-party URLs are referenced in this documentation set and provide additional, related information. Sun is not responsible for the availability of third-party Web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Feedback

Sun Microsystems is interested in improving its documentation and welcomes your comments and suggestions. To share your thoughts, go to `http://docs.sun.com` and click the Send Comments link at the bottom of the page. In the online form provided, include the document title and part number. The part number is a seven-digit or nine-digit number that can be found on the title page of the book or at the top of the document. For example, the title of this book is *Sun Java System Access Manager 7 2005Q4 Federation and SAML Administration Guide*, and the part number is 819-2142.

Documentation, Support, and Training

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

Typographic Conventions

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

TABLE P–2 Typographic Conventions

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

Shell Prompts in Command Examples

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

TABLE P–3 Shell Prompts

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

PART I

The Liberty Alliance Project Specifications and Access Manager

◆ ◆ ◆ CHAPTER 1

Introduction to the Liberty Alliance Project

Sun Java™ System Access Manager implements identity federation, single sign-on (SSO), and web services specifications defined by the Liberty Alliance Project. This introductory chapter explains concepts used in the specifications, and the role of the Liberty Alliance Project in creating identity-based solutions.

This chapter covers the following topics:

Overview of the Liberty Alliance Project

In 2001 Sun Microsystems joined with other major companies to form the *Liberty Alliance Project*. The goals were to define standards for developing identity-based infrastructures, software, and web services, and to promote adoption of these standards. The Liberty Alliance Project does not deliver products or services. It defines frameworks to ensure interoperability between homogeneous products while respecting the privacy and security of identity data.

Note – If you are already familiar with the concepts and protocols developed by the Liberty Alliance Project, go to Chapter 2 for information on how these standards are integrated into Access Manager.

Members of the Liberty Alliance Project

The members of the Liberty Alliance Project include some of the world's most recognized companies, representing products, services and partnerships across a wide spectrum of consumer

and business service providers. Members also include government organizations and technology vendors. For a complete listing of current members, see the Liberty Alliance Project web site.

Note – Only members of the Liberty Alliance Project are allowed to provide feedback on drafts of the specifications although any organization may implement them.

Objectives of the Liberty Alliance Project Specifications

The specifications developed by the Liberty Alliance Project enable individuals and organizations to securely conduct network transactions. The main objectives include:

- Serve as open standards for federated identity management and web services.
- Support and promote permission-based sharing of personal identity attributes.
- Provide a standard for SSO that includes decentralized authentication and authorization for multiple providers.
- Create an open network identity infrastructure that supports all current and emerging *user agents* (also referred to as browsers or wireless browsers).
- Enable consumers to protect their network identity information.

Concept of Identity

In one dictionary, *identity* is defined as "a set of information by which one person is definitively distinguished". This information begins with a document that corroborates a person's name: a birth certificate. Over time, additional information further designates aspects of identity:

- An address
- A telephone number
- One or more diplomas
- A driver's license
- A passport
- Financial institution accounts
- Medical records
- Insurance statements
- Employment records
- Magazine subscriptions
- Utility bills

Each of these individual documents represents data that defines a person's identity as it relates to the enterprise for which the identity was defined. The composite of this data constitutes an overall identity with each specific piece providing a distinguishing characteristic.

Because the Internet is becoming the primary vehicle for the types of interactions represented by this identity-defining information, people are now creating online identities specific to the businesses with which they interact. By defining a user identifier and password, an email address, personal preferences (such as style of music, or opt-in/opt-out marketing decisions) and other information more specific to the particular business (a bank account number or ship-to address), users distinguish themselves from others who use the enterprise's services. This distinguishing information is referred to as a *local identity* because it is specific to the service provider for which it has been set.

Considering the number of service providers for which you can define a local identity, accessing each provider can be a time-consuming and frustrating experiencing. In addition, although most local identities are configured independently (and fragmented across the Internet), it might be useful to connect the information. For example, a user's local identity with a bank could be securely connected to the same user's local identity with a retailer. Federation addresses this issue.

Concept of Federation

Federation is defined as "an association formed by merging several groups or parties". In the Liberty Alliance Project specifications, federation encompasses both *identity federation* and *provider federation*.

Identity Federation

Federation, as it has evolved with regard to the World Wide Web, begins with the notion of identity. Sending and receiving email, checking bank balances, finalizing travel arrangements, accessing utility accounts, and shopping are just a few online services for which a user might define an identity. Now, in order to access the service, the user logs in to the service provider, a networked entity that provides services to other entities.

If a user accesses these services, many user accounts have been configured separately. This virtual phenomenon offers an opportunity to fashion a system for users to federate their disparate service provider identities.

Identity federation allows the user to link, connect, or bind the local identities that have been created for the multiple service providers. The linked local identities, referred to as a *federated identity*, allow the user to log in to one service provider site and click through to an affiliated service provider without having to reauthenticate or reestablish identity.

Provider Federation

The concept of federation as defined by the Liberty Alliance Project begins with a "circle of trust." A *circle of trust* is a group of service providers who contractually agree to exchange authentication information using a Liberty-enabled architecture. Each circle must also include at least one identity provider. An *identity provider* is a service provider that maintains and manages identity data, and provides authentication services.

Note – The establishment of contractual agreements between providers is beyond the scope of this guide. For information, see the Liberty Trust Model Guidelines.

After the contracts and policies defining a circle of trust are in place, the specific protocols, profiles and security mechanisms being used in the deployment are distilled into a metadata document that is exchanged between the members of the circle of trust. Access Manager provides the tools necessary to integrate the metadata and enable the circle technologically as an *authentication domain*. Authentication within this virtual federation is honored by all membered providers of the authentication domain. For more information, see "Authentication Domain" on page 29.

Liberty Alliance Project Concepts

Many of the concepts defined in this section are derived from the specifications discussed in "Liberty Alliance Project Specifications" on page 34.

Account Federation

See "Identity Federation" on page 31.

Affiliation

An *affiliation* is a group of providers formed without regard to a particular authentication domain. An affiliation is formed and maintained by an *affiliation owner*. Members of an affiliation may invoke services either as a member of the affiliation (by virtue of their Affiliation ID) or individually (by virtue of their Provider ID). An *affiliation document* describes a group of providers. See Chapter 3 for more information.

Attribute Provider

An *attribute provider* is a web service that hosts attribute data, for example, an instance of the Liberty Personal Profile Service data service. For more information, see Chapter 6.

Authentication Context

Authentication context refers to information added to a SAML Authentication Assertion regarding details of the technology used for the actual authentication action. This information might include the method of authentication (HTTP Basic or Safeword), the process followed in the issuance of the identity (for example, web self-registration), and any other characteristics that may be relevant to the SAML assertion consumer. The following XML example describes a user having authenticated with a password over an SSL-protected session:

EXAMPLE 1–1 XML Sample Defining Authentication Context

```
<?xml version="1.0" encoding="UTF-8" ?>
<AuthenticationContextStatement>
            <AuthenticationMethod>
                <PrincipalAuthenticationMethod>
                   <Password>
                      <Length min="3"/>
                   </Password>
                </PrincipalAuthenticationMethod>
                <AuthenticatorTransportProtocol>
                   <SSL/>
                </AuthenticatorTransportProtocol>
            </AuthenticationMethod>
<AuthenticationContextStatement>
```

Authentication Domain

An *authentication domain* is a federation of service providers (with at least one identity provider) that is configured technologically. The providers interact using the Liberty Alliance Project specifications. The term *authentication domain* does not encompass the prerequisite business agreements established between providers in a circle of trust. After the circle of trust is established, an authentication domain can be configured and single sign-on can be enabled.

Note – An authentication domain is not a domain in the Domain Name System (DNS) sense of the word.

Circle of Trust

See "Provider Federation" on page 27.

Client

A *client* is the role that any system entity assumes when making a request of another system entity. In this scenario, the system entity to which the request is made is called a *server* as discussed in "Server" on page 32.

Common Domain

If an authentication domain has more than one identity provider, the service providers need a way to determine which identity provider is used by the principal (as discussed in "Principal" on page 32). Because this function must work across any number of DNS domains, the Liberty approach is to

create one domain that is common to all identity and service providers in the authentication domain. This predetermined domain is called the *common domain*. Within the common domain, when a principal has been authenticated to a service provider, the identity provider writes a *common domain cookie* that stores the principal's identity provider. When the principal attempts to access another service provider within the authentication domain, the service provider reads the common domain cookie and the request is forwarded to the correct identity provider. See Chapter 4 for more information.

Defederation

See "Federation Termination" on page 30.

Federation

See "Concept of Federation" on page 27.

Federation Cookie

A *federation cookie* called `fedCookie` is implemented by Access Manager. It can have a value of `yes` or `no`, based on the principal's federation status. For information on how a federation cookie is used, see "Process of Federation" on page 64 in Chapter 3.

Note – The concept of a *federation cookie* was developed for Access Manager and is not a defined part of the Liberty Alliance Project specifications. The definition is placed here for information only.

Federated Identity

A *federated identity* refers to the consolidated account information that a user has provided to service providers. Personal data, authentication information, buying habits and history, and shopping preferences are examples of user account information. The information is administered by the user, and can be securely shared with other service providers.

Federation Termination

Users can terminate their federations. *Federation termination*, or *defederation*), cancels identity federations established between the user's identity provider and service provider accounts.

Identity

See "Concept of Identity" on page 26.

Identity Federation

Identity federation occurs when a user chooses to unite distinct service provider accounts with one or more identity provider accounts. A user retains the individual account information with each provider while, simultaneously, establishing a link that allows the exchange of authentication information between them. For more information, see "Concept of Federation" on page 27.

Identity Provider

An *identity provider* is a service provider that specializes in providing authentication services. As the administrating service for authentication, an identity provider also maintains and manages identity information. Authentication by an identity provider is honored by all service providers with whom the identity provider is affiliated. This term is used when defining an entity of this sort specific to the Liberty Identity Federation Framework as discussed in "Liberty Identity Federation Framework" on page 34.

Identity Service

An *identity service* (also referred to as a *data service*) is a web service that acts on a resource to retrieve, update, or perform some action on data attributes related to a principal (an *identity*). For example, an identity service might be a corporate phone book or calendar service. For more information, see Chapter 6.

Liberty-Enabled Client

A *Liberty-enabled client* is a client that has, or knows how to obtain, information about the identity provider that a principal will use to authenticate to a service provider.

Liberty-Enabled Proxy

A *Liberty-enabled proxy* is an HTTP proxy that emulates a Liberty-enabled client.

Name Identifier

To help preserve anonymity when identity information is exchanged between identity and service providers, an arbitrary name identifier is used. A *name identifier* is a randomly generated character string that is assigned to a principal and used to facilitate account linking at the identity provider and service provider sites. This pseudonym allows all providers to identify a principal without knowing the user's actual identity. The name identifier has meaning only in the context of the relationship between providers.

Principal

A *principal* is an entity that can acquire a federated identity, that is capable of making decisions, and has authenticated actions done on its behalf. Examples of principals include an individual user, a group of individuals, a corporation, other legal entities, or a component of the Liberty architecture.

Profile

A Liberty-based *profile* defines the combination of a message's content and its transport mechanisms for a user agent.

Provider Federation

See "Concept of Federation" on page 27.

Pseudonym

See "Name Identifier" on page 31.

Receiver

A *receiver* is the role of a system entity when it receives a message sent by another system entity. In this scenario, the system entity from which the message is received is called a *sender* as discussed in "Sender" on page 32.

Resource Offering

In a discovery service, a *resource offering* defines associations between a piece of identity data and the service instance that provides access to it. See Chapter 7.

Sender

A *sender* is the role donned by a system entity when it constructs and sends a message to another system entity. In this scenario, the system entity from which the message is received is called a *receiver* as discussed in "Receiver" on page 32.

Server

A *server* is the role that any system entity assumes when providing a service in response to a request from another system entity. In this scenario, the system entity from which the request is received is called a client as discussed in "Client" on page 29.

Note – In order to provide a service to clients, a server will often be both a sender and a receiver.

Service Provider

A *service provider* is a commercial or not-for-profit organization that offers web-based services to a principal. This broad category can include Internet portals, retailers, transportation providers, financial institutions, entertainment companies, libraries, universities, and governmental agencies. This term is used when defining an entity of this sort specific to the Liberty Identity Federation Framework as discussed in "Liberty Identity Federation Framework" on page 34.

Single Logout

A *single logout* occurs when a user logs out of an identity provider or a service provider. By logging out of one provider, the user is logged out of all service providers or identity providers in that authentication domain.

Single Sign-On

Single sign-on is established when a user with a federated identity authenticates to an identity provider. If the user has previously opted-in for federation, access to affiliated service providers without having to re-authenticate is available.

Trusted Provider

A *trusted provider* is a generic term for one of a group of service and identity providers in an authentication domain. A user can transact and communicate with trusted providers in a secure environment.

Web Service Consumer

A *web service consumer* invokes the operations that a web service provides by making a request to a web service provider. This term is used when defining an entity of this sort specific to the Liberty Identity Web Services Framework as discussed in "Liberty Identity Web Services Framework" on page 39.

Web Service Provider

A *web service provider* implements a web service based on a request from a web service consumer. This term is used when defining an entity of this sort specific to the Liberty Identity Web Services Framework as discussed in "Liberty Identity Web Services Framework" on page 39.

Note – A web service provider may run on the same Java virtual machine as the web service consumer that is using it.

Liberty Alliance Project Specifications

The Liberty Alliance Project develops and delivers specifications that enable federated network identity management. Using web redirection and open-source technologies such as SOAP and XML, they enable distributed, cross-domain interactions. The specifications are divided into the following components:

- "Liberty Identity Federation Framework" on page 34
- "Liberty Identity Web Services Framework" on page 39
- "Liberty Identity Service Interface Specifications" on page 41

There are also many support documents in the specifications, including a metadata service protocol, reverse HTTP bindings, a glossary, and schema files. For more information on all of the documents, see the Liberty Alliance Project web site.

Liberty Identity Federation Framework

The *Liberty Identity Federation Framework* (Liberty ID-FF) defines a set of protocols, bindings, and profiles that provides a solution for identity federation, cross-domain authentication, and session management. This framework can be used to create a new identity management system or to develop one in conjunction with legacy systems. The Liberty ID-FF is designed to work with heterogeneous platforms, various networking devices (including personal computers, mobile phones, and personal digital assistants), and emerging technologies. The following figure shows the subjects involved in a Liberty ID-FF implementation.

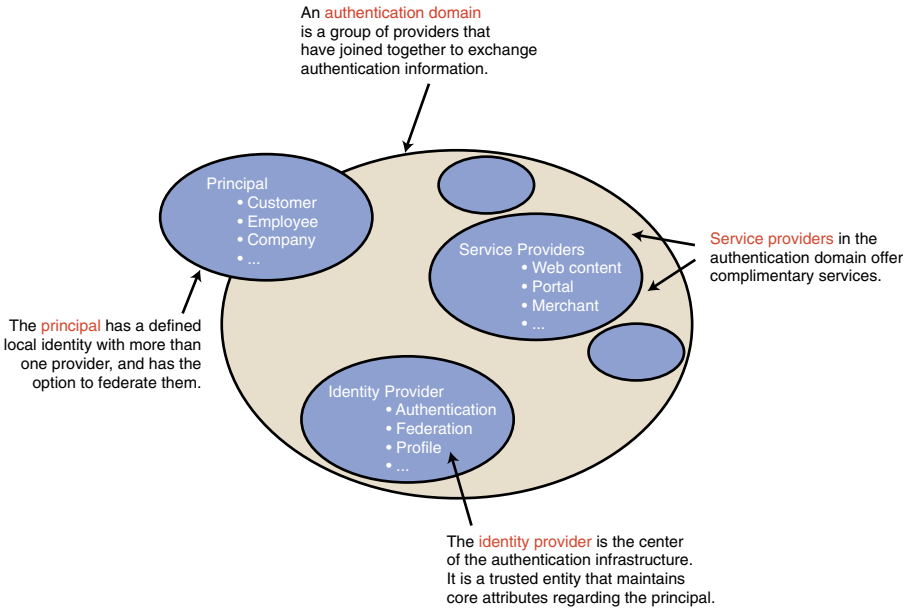

FIGURE 1–1 Subjects Involved in a Liberty ID-FF Implementation

- A *principal* can have a defined local identity with more than one provider, and it has the option to federate the identities. The principal might be an individual user, a group of individuals, a corporation, or a component of the Liberty architecture.
- A *service provider* is a commercial or not-for-profit organization that offers a web-based service such as a news portal, a financial repository, or retail outlet.
- An *identity provider* is a service provider that stores identity profiles and offers incentives to other service providers for the prerogative of federating their user identities. Identity providers might also offer services above and beyond those related to identity profile storage.
- To support identity federation, both service and identity providers must join together into an *authentication domain*. An authentication domain must contain at least one identity provider and at least two service providers. One organization may be both an identity provider and a service provider.

Organizations in an authentication domain must first write operational agreements to define their relationships in a circle of trust. An *operational agreement* is a contract between organizations that defines how the circle will work. For more information, see "Authentication Domain" on page 29 and "Provider Federation" on page 27.

Liberty ID-FF Protocols and Schema

The *Liberty ID-FF Protocols and Schema Specifications* defines these abstract protocols:

- "Single Sign-On and Federation Protocol" on page 36
- "Name Registration Protocol" on page 37

Following are short explanations of each protocol. More detailed information can be found in the *Liberty ID-FF Protocols and Schema Specifications*.

Single Sign-On and Federation Protocol

The *Single Sign-On and Federation Protocol* defines a request and response protocol by which a principal is able to authenticate to one or more service providers and *federate* (or link) configured identities. A service provider issues a request for authentication to an identity provider. The identity provider responds with a message that contains authentication information, or an *artifact* that points to authentication information. The identity provider can also federate the principal's identity (configured at the identity provider level) with the principal's identity (configured at the service provider level).

Note – Under certain conditions, an identity provider may issue an authentication response to a service provider without having received an authentication request.

The *Single Sign-On and Federation Protocol* also defines controls that allow for the following behaviors:

- **Account federation.** A principal can choose to federate a configured identity at the identity provider site with a configured identity at the service provider site.
- **Account handle.** An identity provider can issue an anonymous, temporary identifier to refer to a particular principal during communication with a service provider. This identifier is used to obtain information for or about the principal during federation (with the principal's consent). The account handle is generated by the identity provider during federation. This account handle is not to be confused with the handle that can be generated by the service provider after federation using the *Name Registration Protocol* as discussed in "Name Registration Protocol" on page 37.
- **Affiliation federation.** Federation based on group affiliation can be enabled in an authentication request. If enabled, it indicates that the requester is acting as a member of the specified affiliation group. Federations are then established and resolved based on the affiliation, not the requesting provider. The process allows for a unique identifier that represents the affiliation.
- **Authentication context.** A service provider can choose the type and level of authentication that should be used when a principal logs in.
- **Authentication credentials.** A principal can be prompted to authenticate with a user name and password, for example, at the behest of the service provider.
- **Dynamic identity provider proxying.** One identity provider might be asked to authenticate a principal that has already been authenticated by a second identity provider. In this case, the first identity provider may request authentication information from the second identity provider on behalf of the service provider. Proxy behavior can be controlled by indicating a list of preferred identity providers, and a value that defines the maximum number of proxy steps that can be

taken. Proxy behavior is defined locally by the proxying identity provider, although a service provider controls whether or not to proxy. For more information, see "Dynamic Identity Provider Proxying" on page 97.

- **Identity provider introduction.** When an authentication domain has more than one identity provider, a service provider can use this feature to determine which identity provider a principal is using.
- **Message exchange profiles.** The authentication request defines how messages are exchanged between identity providers and service providers. The particular transfer and messaging protocol used in the exchange (such as HTTP or SOAP) are specified in *profiles*. Two of these profiles are:
  - The Liberty Artifact profile relies on Security Assertion Markup Language (SAML) artifacts and assertions to relay authentication information.
  - The Liberty Browser POST profile relies on an HTML form to communicate authentication information between providers.
- **One-time federation.** The ability to federate for one session only can be enabled in an authentication request. This feature is useful for service providers with no user accounts, for principals who want to act anonymously, or for dynamically created user accounts. It allows for one-time federation, rather than a one-time name identifier for a session.

Name Registration Protocol

The optional *Name Registration Protocol* is used by the service provider to create its own opaque handle to identify a principal when communicating with the identity provider.

Note – The handle discussed in this section is not related to the opaque handle that is generated by the identity provider during federation as defined in "Single Sign-On and Federation Protocol" on page 36. The Name Registration Protocol can, however, be used by the identity provider to change the opaque handle that it registered with the service provider during initial federation.

Federation Termination Notification Protocol

The *Federation Termination Notification Protocol* defines how the identity provider or the service provider notifies the other provider when a principal has terminated identity federation. The notification is a one-way, asynchronous message which states one of the following:

- The service provider will no longer accept authentication information regarding the particular user.
- The identity provider will no longer provide authentication information regarding the particular user.

Single Logout Protocol

The *Single Logout Protocol* defines how providers notify each other of logout events. This message exchange protocol is used to terminate all sessions when a logout occurs at the service provider or

identity provider. The particular transfer and messaging protocol used in the exchange (such as HTTP or SOAP) are specified in *profiles*. Two of these profiles are:

- The SOAP/HTTP-based profile relies on asynchronous SOAP over HTTP messaging calls between providers.
- The HTTP Redirect-based profile relies on HTTP redirects between providers.

Name Identifier Mapping Protocol

The *Name Identifier Mapping Protocol* defines how service providers can obtain name identifiers that are assigned to a principal that has federated in the name space of a different service provider. When a principal authenticated to one service provider requests access to a second service provider site, the second service provider can use this protocol to obtain the name identifier. The protocol allows the second service provider to communicate with the first service provider about the principal even though no identity federation for the principal exists between them.

Liberty ID-FF Bindings and Profiles

The *Liberty ID-FF Bindings and Profiles Specification* defines the bindings and profiles for the Liberty protocols and messages sent to HTTP-based communication frameworks. This specification relies on the core SAML framework. For example, the *Name Identifier Encryption Profile* permits a principal's name identifier to be encrypted so that only the provider possessing the decryption key can realize the identity. The encrypted identifier is a different value when requested by different providers. Using different values reduces the chance for correlation of the encrypted value across multiple logical transactions. For more information about the *Name Identifier Encryption Profile* and the specification in general, see the *Liberty ID-FF Bindings and Profiles Specification*.

Additional Liberty ID-FF Documents

For additional information about the Liberty ID-FF specifications, see the following documents.

- *Liberty ID-FF 1.2 Architecture Overview*

  Provides an architectural description of the Liberty ID-FF framework as well as policy, security, and technical notes.

- *Liberty ID-FF 1.2 Implementation Guidelines*

  Provides guidance and checklists for implementing a Liberty-enabled environment using the Liberty ID-FF specifications.

- *Liberty ID-FF 1.2 Static Conformance Requirements*

  Defines what features are mandatory and optional for implementations conforming to this version of the Liberty ID-FF specifications.

Liberty Identity Web Services Framework

The Liberty ID-FF defines how to implement single sign-on and identity federation to solve problems related to network identity. The *Liberty Identity Web Services Framework* (Liberty ID-WSF) builds on this by providing specifications to develop web services that retrieve, update, or perform an action on identity data in a federated network environment. The specifications outline the technical components needed to build web services that operate with identity data, such as a calendar service, a wallet service, or an alert service. A scenario that implements these specifications includes the following subjects:

- A *web service consumer* (WSC) invokes the operations that a web service provides by making a request to a web service provider.
- A *web service provider* (WSP) implements a web service based on a request from a web service consumer.

Web services are the basis of distributed computing across the Internet. A WSC locates a web service and invokes the operations the web service provides. The WSP is the application that implements a web service. The web service can be on the same Java virtual machine as the WSC, or it can be thousands of miles away. When a WSC needs to retrieve identity attributes from a WSP, the WSC must first contact a discovery service to locate where the particular attributes are stored. When this information is returned, the WSC then contacts the WSP (for example, a personal profile service) to retrieve the necessary attributes.

For more information about the process between a WSC and WSP, see "Discovery Service Process" on page 136.

Liberty ID-WSF Specifications

The Liberty ID-WSF includes these specifications:

SOAP Binding Specification

The *Liberty ID-WSF SOAP Binding Specification* provides a transport layer for handling SOAP messages. It defines SOAP header blocks and processing rules that enable the invocation of identity services using SOAP requests and responses. It also specifies how to 1) configure messages for optimum message correlation, assuring the relationship between a SOAP request and its response, 2) consent claims (permission to perform a certain action), and 3) usage directives (data handling policies). For more information, see the *Liberty ID-WSF SOAP Binding Specification*.

Discovery Service Specification

The *Liberty ID-WSF Discovery Service Specification* defines a framework that enables a client to locate the appropriate web service for retrieving, updating, or modifying a particular piece of identity data. Typically, there are one or more services on a network that allow entities to perform an action on identity data. To keep track of these services or to know which can be trusted, clients require a discovery service. A *discovery service* is essentially a web service interface for a registry of resource offerings. A *resource offering* defines an association between a piece of identity data and the service instance that provides access to the data. A common use case is when a personal profile or calendar data is placed within a discovery resource so that the data can be located by other entities. For more information, see the *Liberty ID-WSF Discovery Service Specification*.

Security Mechanisms Specification

The *Liberty ID-WSF Security Mechanisms Specification* describes the requirements for securing authorization decisions that are sent for the discovery and use of identity services. The specified mechanisms provide for authentication, signing, and encryption operations to ensure integrity and confidentiality of the messages. For more information, see the *Liberty ID-WSF Security Mechanisms Specification*.

Data Services Template Specification

The *Liberty ID-WSF Data Services Template Specification* defines how to query and modify the identity data attributes that are stored in a *data service* (a web service that holds data). The specification also provides common attributes for data services. For more information, see the *Liberty ID-WSF Data Services Template Specification*.

Interaction Service Specification

The *Liberty ID-WSF Interaction Service Specification* provides communication protocols for identity services to obtain permission from a principal (or someone who owns a resource on behalf of that principal) that allows the service to share the principal's identity data with requesting services. For more information, see the *Liberty ID-WSF Interaction Service Specification*.

Authentication Service Specification

The *Liberty ID-WSF Authentication Service Specification* defines how to authenticate parties communicating via SOAP-based messages. It leverages widely used authentication services and mechanisms, and facilitates selection of these services and mechanisms at deployment time. The specification defines the following:

- An authentication protocol based on the Simple Authentication and Security Layer (SASL).
- An authentication service that Liberty-enabled clients can use to authenticate with identity providers.
- A single sign-on service that Liberty-enabled providers can use to interact with each other.

The specification also defines an identity-based authentication security token service, complementing the more general security token service as discussed in the section, "Discovery Service Specification" on page 40. For more information, see the *Liberty ID-WSF Authentication Service Specification*.

Client Profiles Specification

The *Liberty ID-WSF Client Profiles Specification* describes the requirements for Liberty-enabled clients that interact with the SOAP-based Authentication Service. Client profiles can enable browsers to perform an active role in transactions, in addition to the functions of a standard browser. For more information, see the *Liberty ID-WSF Client Profiles Specification*.

Additional Liberty ID-WSF Documents

For additional information about the Liberty ID-WSF specifications, see the following documents:

- *Liberty ID-WSF Architecture Overview*

  Provides an architectural description of the Liberty ID-WSF framework including basic usage scenarios. It also highlights how the Liberty ID-WSF interacts with an identity management framework (such as the Liberty ID-FF).

- *Liberty ID-WSF Security and Privacy Overview*

  Provides an overview of security and privacy issues in the Liberty ID-WSF.

- *Liberty ID-WSF Implementation Guidelines*

  Provides guidelines on how the Liberty ID-WSF specifications should be implemented.

Liberty Identity Service Interface Specifications

The *Liberty Identity Service Interface Specifications* (Liberty ID-SIS) are for building identity-based web services. Included in the Liberty ID-SIS are the following:

- "Liberty ID-SIS Personal Profile Service Specification" on page 41
- "Liberty ID-SIS Employee Profile Service Specification" on page 42
- "Additional Liberty ID-SIS Service Specifications" on page 42

Liberty ID-SIS Personal Profile Service Specification

The *Liberty ID-SIS Personal Profile Service Specification* defines an identity-based web service that keeps, updates, and offers identity data regarding a user. This service queries and updates of attribute data and incorporates mechanisms for access control and conveying data validation information and usage directives from other specifications. A shopping portal that offers information such as the principal's account number and shopping preferences is an example of a personal profile service. For more information, see the *Liberty ID-SIS Personal Profile Service Specification*.

Liberty ID-SIS Employee Profile Service Specification

The *Liberty ID-SIS Employee Profile Service Specification* defines an identity-based web service that keeps, updates, and offers profile information regarding a user's workplace. An online corporate phone book that provides an employee name, office building location, and telephone extension number is an example of an employee profile service. For more information, see the *Liberty ID-SIS Employee Profile Service Specification*.

Additional Liberty ID-SIS Service Specifications

The Liberty Alliance Project defines several other Liberty ID-SIS that are not discussed in this section, including a contact book, a geolocation service, and a presence service. For more information on these services, see the documentation in the *Liberty ID-SIS*.

Deploying a Liberty-based System

To build a successful Liberty-based implementation, consider the issues described in this section. At the minimum, a Liberty-compliant identity server is needed to process Liberty-based requests and responses.

Assess the Qualifications of Your IT Staff

Although the specifications are aimed at large organizations, small and medium-sized companies with an experienced IT staff can also roll out a federated identity system. The specifications are complex and require several areas of expertise, including web services development, XML, networking, and security.

Clean Up Directory Data

The specifications do not specify where to store identity data. Purge your data store of old identity profiles, consolidate multiple (or delete duplicated) identity profiles, and ensure that privileges are assigned correctly.

Tip – Identity providers must enforce strict regulations regarding passwords. A stolen identity can be abused across multiple sites in a federated system.

Draft Business Agreements

The specifications assume existing trust relationships between members in a circle of trust. This trust is defined through business arrangements or contracts that describe the technical, operational, and legal responsibilities of each party and the consequences for not completing them. When defined, a

Liberty trust relationship means that one organization trusts another's user authentication decisions. That trust among members enables a user to log in at one site and access another site as well. Ensure that these agreements are in force before going live with a Liberty-compliant system including configured authentication domains.

◆ ◆ ◆ CHAPTER 2

Implementation of the Liberty Alliance Project Specifications

Sun Java System Access Manager contains the Sun Microsystems implementation of the Liberty Alliance Project specifications. This chapter provides an overview of how these specifications have been implemented.

This chapter covers the following topics:

Overview

Sun Java System Access Manager is a software product that helps organizations manage secure access to the resources and web applications within their intranet and across the Internet. The initial release of Access Manager implemented the *Liberty Identity Federation Framework* (Liberty ID-FF) specifications, focusing on account federation, authentication domains, and single sign-on.

Subsequent releases of Access Manager added new features as defined in Version 1.2 of the Liberty ID-FF specifications as well as the Version 1.0 specifications of the *Liberty Identity Web Services Framework* (Liberty ID-WSF). These web services include a framework for retrieving and updating *identity data* which consists of attributes stored in identity-based service providers across the Internet. Also provided are an application programming interface (API) for communication between identity providers and service providers.

This version of Access Manager provides additional functionality based on the Liberty Alliance Project specifications. For example, Access Manager 7 provides the ability to bulk-federate user accounts to applications that are outsourced to business partners. It also provides the ability to map configured roles between the identity provider and the service provider. More specifically, Access Manager 7 2005Q4 supports the Liberty ID-FF 1.1 and 1.2, the Liberty ID-WSF 1.0, and the *Liberty Identity Services Interface Specifications* (Liberty ID-SIS) 1.0.

Liberty Use Cases

Identity data consists of all the information that companies maintain about individual customers, corporate partners, and employees. Federating sources of identity data allows for accessing, transporting, sharing, and managing the data between partnered organizations and applications without weakening existing security safeguards. There are many ways to use Access Manager and its Liberty-based implementations to federate sources of identity data. The following sections explain just a few ways.

Unified Access to Intranet Resources

Many corporations provide access to outsourced human resources services, such as health benefits and 401(k) plans. The corporate intranet offers central access to these services, but employees have to log in and authenticate themselves every time they access each service. Employees might not want to share the same profile and password with both their 401(k) provider and their health care provider. Federation of identity data can provide seamless integration of web resources across multiple security domains within the same enterprise, allowing for employee ease-of-use and control.

Integrated Partner Networks

Enterprises can construct a network of partnered services for securely exchanging customer account information, transaction data, and credentials through a set of interoperable web services. Federation among partner networks allows identities to share key pieces of their respective data without sharing control. For example, logging in to one web site that represents an *authentication domain* consisting of an airline, a car rental company, and a hotel chain allows an identity to make travel plans even if one of the sites does not contain an identity data store.

Sample Use Case Process

Using a cell phone, a principal is able to access a ring-tone vendor's site. Due to implementation of single sign-on, the ring-tone vendor recognizes the principal from the cell-phone provider's authentication. This allows the principal to purchase ring tones by interacting with the user's bank for payment. The following figure illustrates the process of requesting a service and being authenticated for access. It assumes the following:

- *MyWireless* is a cellular service provider and an identity provider in a federation framework that contains access to the discovery service in a web services framework.
- *MyRingtones* is a service provider in a federation framework that also acts as a web service consumer (WSC) in a web services framework. It sells ringtones for use with cellular phones.
- *MyBank* is a web service provider (WSP) in a web services framework. Linking *MyBank* to *My Ringtones* offers the opportunity for seamless purchases.

Note – The same web service can act as a different entity in different scenarios.

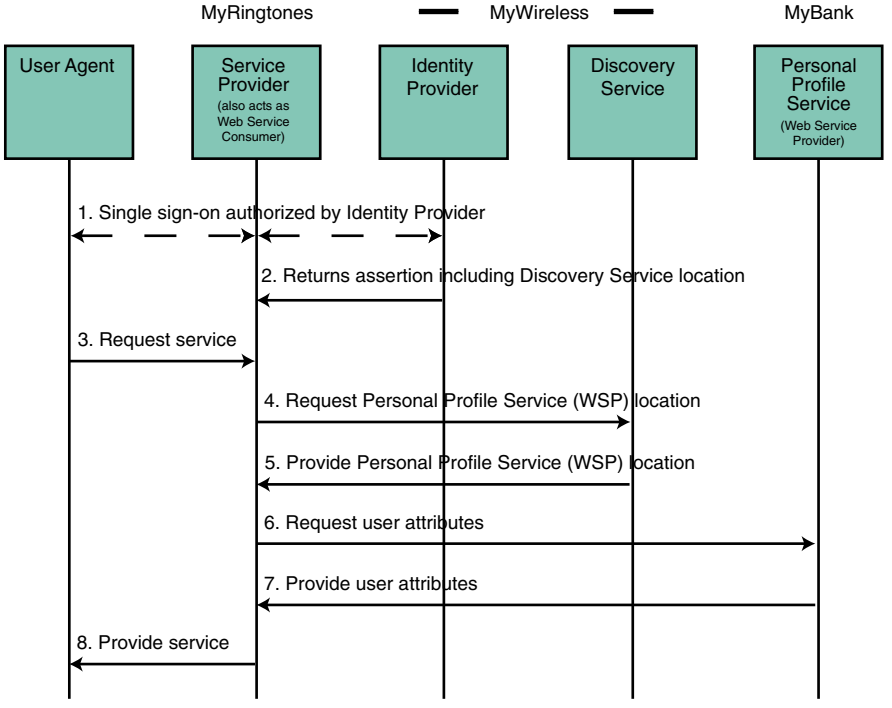

FIGURE 2–1 Process in a Liberty-enabled Use Case

The user attempts to access *MyRingtones* and, after being prompted for credentials stored in *MyBank*, receives authorization through *MyWireless*. Single sign-on is accomplished in the back end. The entire process is based on implementations of the Liberty ID-FF, Liberty ID-WSF, and Liberty ID-SIS specifications.

Liberty Alliance Project Architecture in Access Manager

The figure below shows the architecture of the Access Manager features that are based on the Liberty Alliance Project specifications. These features leverage existing Access Manager services including policy, service management, session management, and auditing.

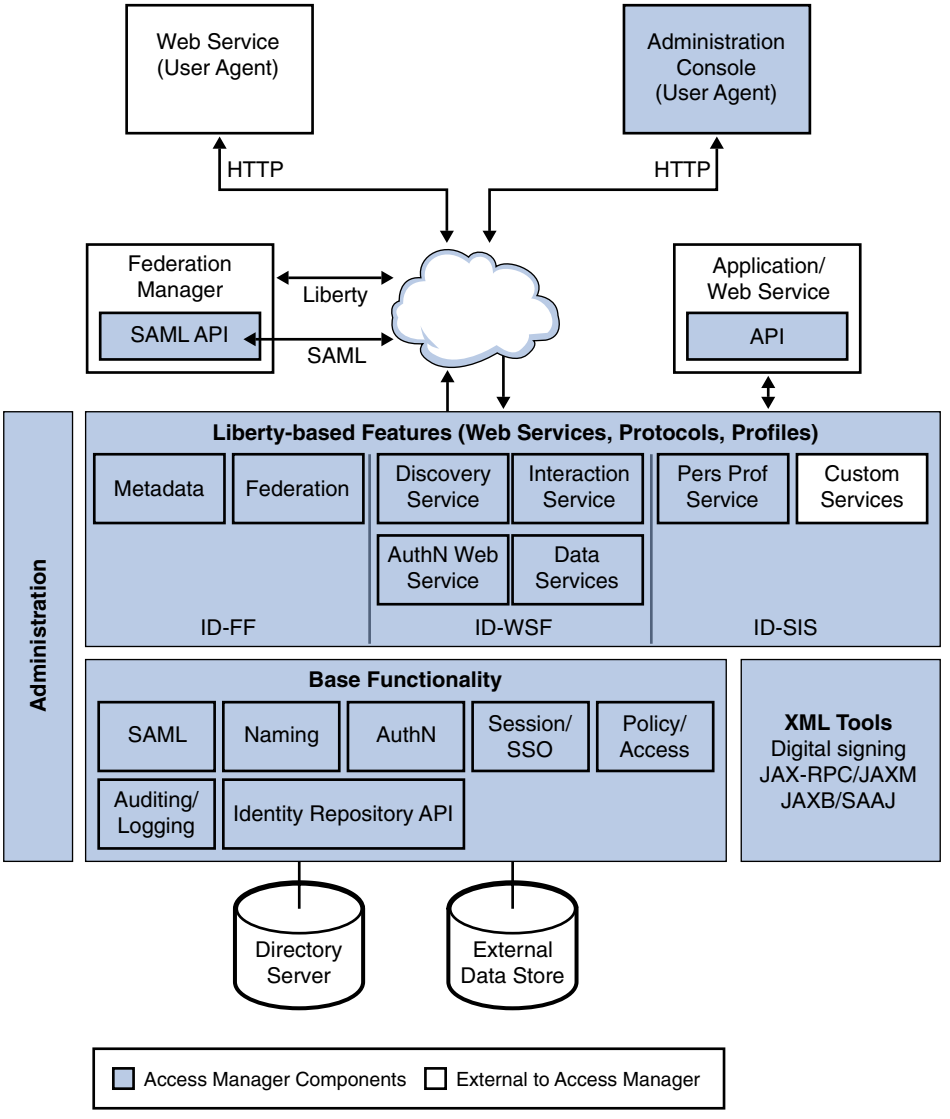

FIGURE 2–2 Liberty-based Architecture of Access Manager

Note – For a complete architectural overview of Access Manager, see the *Sun Java System Access Manager 7 2005Q4 Technical Overview*.

Accessing the Liberty Alliance Project Features

Access Manager is installed with a set of default Liberty-based web services. They, the larger Federation component, application programming interfaces, and the Security Assertion Markup Language (SAML) are introduced in the following sections.

Federation in Access Manager

The Federation component of Access Manager provides an interface for creating, modifying, and deleting authentication domains and service and identity providers (both remote and hosted types) for a federated model. The web interface for the Liberty ID-FF in Access Manager is accessible from the Federation tab in the Access Manager Console, as shown in the following figure.

FIGURE 2–3 Federation Interface in Access Manager Console

The following steps illustrate the process for creating a federation model using Access Manager:

1. Create an authentication domain.
2. Configure one or more hosted providers that belong to the authentication domain.
3. Configure one or more remote providers that belong to the authentication domain, and include the metadata for the remote providers.
4. Establish the trusted partnership between the providers. A hosted provider can choose to trust a subset of providers, either hosted or remote, that belong to the same authentication domain.

Liberty-based Web Services in Access Manager

Liberty-based web services are those based on specifications in the Liberty ID-WSF and the Liberty ID-SIS. They are accessible from the Access Manager Console by clicking the Web Services tab. The implemented web services include:

The following diagram illustrates how the different web service specifications have been implemented.

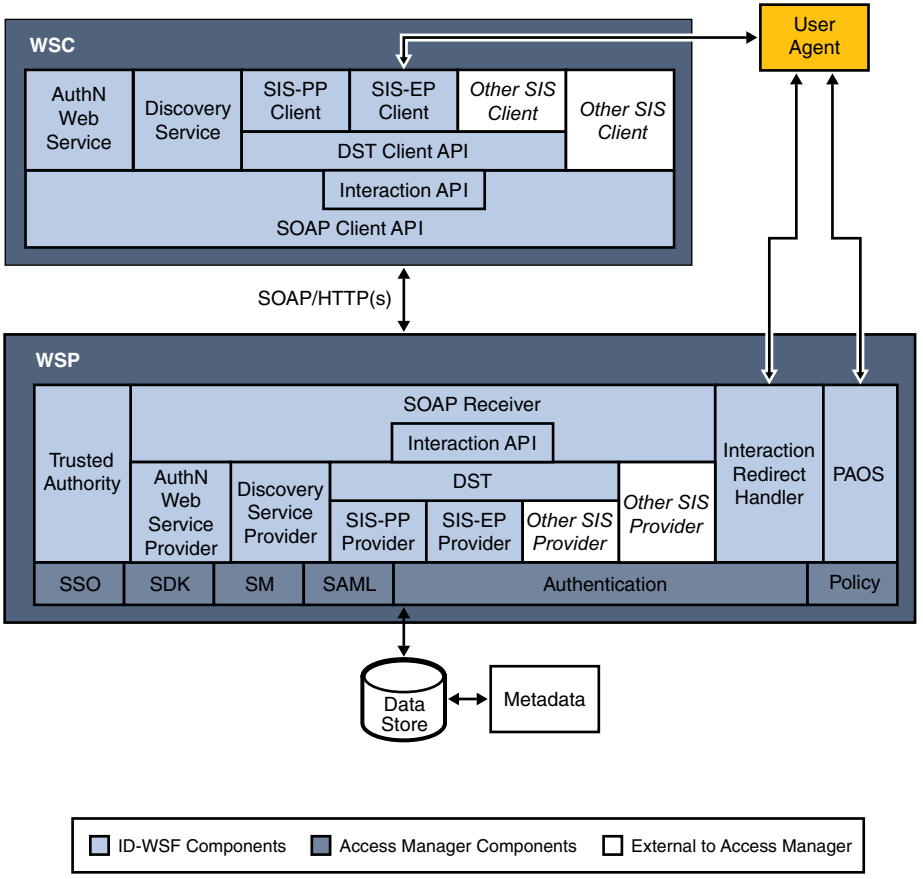
WSC
AuthN Web Service
Discovery Service
SIS-PP Client
SIS-EP Client
Other SIS Client
Other SIS Client
DST Client API
Interaction API
SOAP Client API
User Agent
SOAP/HTTP(s)
WSP
Trusted Authority
SOAP Receiver
Interaction API
AuthN Web Service Provider
Discovery Service Provider
DST
SIS-PP Provider
SIS-EP Provider
Other SIS Provider
Other SIS Provider
Interaction Redirect Handler
PAOS
SSO
SDK
SM
SAML
Authentication
Policy
Data Store
Metadata
ID-WSF Components
Access Manager Components
External to Access Manager

FIGURE 2–4 Architecture of Liberty-based Web Services

The web interface for the Liberty ID-WSF in Access Manager is accessible from the Web Services tab in the Access Manager Console, as shown in the following figure.

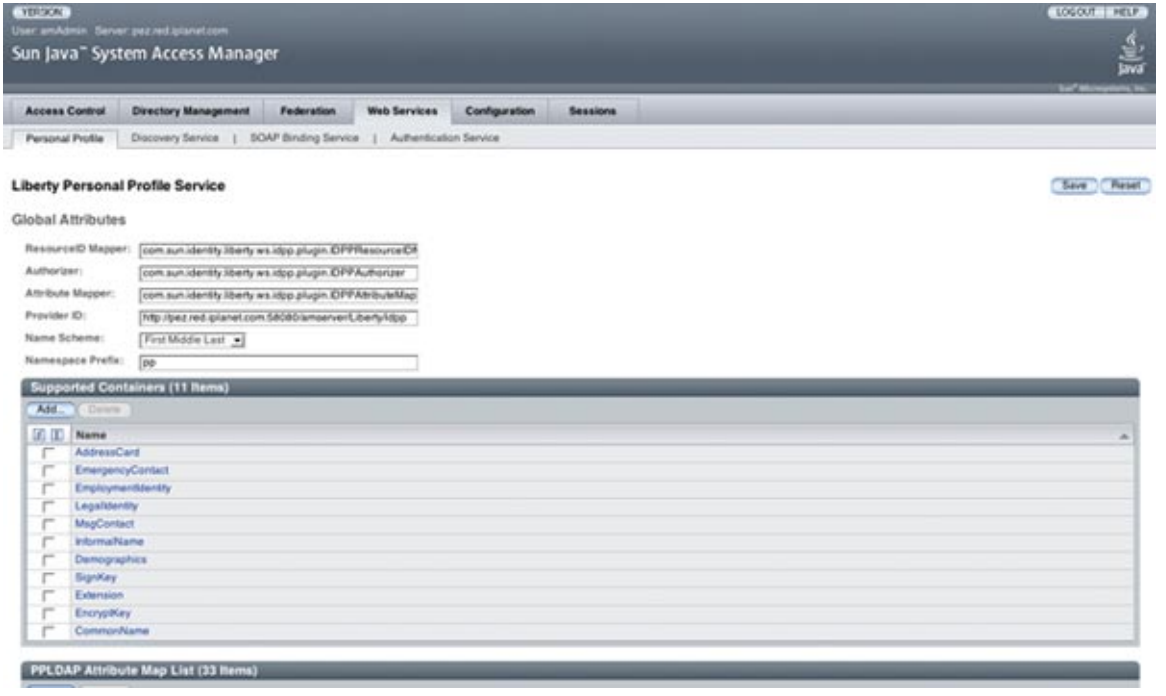

FIGURE 2–5 Web Services Interface in Access Manager Console

Liberty Personal Profile Service

The Liberty Personal Profile Service is a data service that supports storing and modifying a principal's identity attributes. Identity attributes might include information such as first name, last name, home address, and emergency contact information. The Liberty Personal Profile Service is queried or updated by a WSC acting on behalf of the principal. For more information, see Chapter 6.

Discovery Service

The Discovery Service is a web service that allows a requesting entity, such as a service provider, to dynamically determine a principal's registered attribute provider. Typically, a service provider queries the Discovery Service, which responds by providing a resource offering that describes the requested attribute provider. (A *resource offering* defines associations between a piece of identity data and the service instance that provides access to the data.) The implementation of the Discovery Service includes Java and web-based interfaces. For more information, see Chapter 7.

Note – By definition, a discoverable service is assigned a service type Uniform Resource Identifier (URI), allowing the service to be registered in Discovery Service instances. The service type URI is typically defined in the Web Service Definition Language (WSDL) file that defines the service.

SOAP Binding Service

The SOAP Binding Service is a set of Java APIs used by the developer of a Liberty-enabled identity service. The APIs are used to send and receive identity-based messages using SOAP, an XML-based messaging protocol. For more information, see Chapter 8.

Authentication Web Service

The Authentication Web Service provides web service-based authentication to a WSC, allowing the WSC to obtain security tokens for further interactions with other services at the same provider. These other services may include a discovery service or single sign-on service. The Authentication Web Service is for service-to-service (nonuser) authentication. For more information, see Chapter 5.

Note – Do not confuse the Liberty-based Authentication Web Service with the proprietary Access Manager Authentication Service discussed in the *Sun Java System Access Manager 7 2005Q4 Administration Guide*.

Liberty-based Application Programming Interfaces

A number of the Liberty-based web services specifications have also been implemented in the back end of Access Manager as APIs. The services include the Interaction Service and PAOS binding. The following table summarizes the public APIs. They can be used to deploy Liberty-enabled components or extend the core services.

TABLE 2–1 Public Interfaces

Package Name	Description
`com.sun.identity.liberty.ws.authnsvc`	Provides classes to manage the Authentication Web Service. See Chapter 5.
`com.sun.identity.liberty.ws.authnsvc.mechanism`	Provides an interface to process incoming Simple Authentication and Security Layer (SASL) requests and generate SASL responses for the different SASL mechanisms. See Chapter 5.
`com.sun.identity.liberty.ws.authnsvc.protocol`	Provides classes to manage Authentication Web Service protocol. See Chapter 5.

TABLE 2–1 Public Interfaces *(Continued)*

Package Name	Description
`com.sun.identity.liberty.ws.common`	Defines common classes that are used by many of the Access Manager Liberty-based web service components. See "Common Service Interfaces" on page 197 of this chapter.
`com.sun.identity.liberty.ws.common.wsse`	Provides an interface to parse and create a X.509 Certificate Token Profile. See "Common Service Interfaces" on page 197 of this chapter.
`com.sun.identity.liberty.ws.disco`	Provides interfaces to manage the Discovery Service. See Chapter 7.
`com.sun.identity.liberty.ws.disco.plugins`	Provides a plugin interface for the Discovery Service. See Chapter 7.
`com.sun.identity.liberty.ws.dst`	Provides classes to implement an identity service. See Chapter 6 for information about services built using this API.
`com.sun.identity.liberty.ws.dst.service`	Provides a handler class that can be used by any generic identity data service. See Chapter 6 for information about data services.
`com.sun.identity.liberty.ws.interaction`	Provides classes to support the Interaction RequestRedirect Profile. See the section on the "Interaction Service" on page 201 for information on this profile.
`com.sun.identity.liberty.ws.interfaces`	Provides interfaces that are common to all Access Manager Liberty-based web service components. See Chapter 7 and Chapter 6 for information about default implementations. See the section on "Common Service Interfaces" on page 197 for more general information.
`com.sun.identity.liberty.ws.paos`	Provides classes for web applications to construct and process PAOS requests and responses. See "PAOS Binding" on page 203 of this chapter.
`com.sun.identity.liberty.ws.security`	Provides an interface to manage Liberty-based web service security mechanisms. See "Common Security API" on page 199 of this chapter.
`com.sun.identity.liberty.ws.soapbinding`	Provides classes to construct SOAP requests and responses and to change the contact point for the SOAP binding. See Chapter 8.
`com.sun.identity.saml`	Provides a service provider interface (SPI) in which proprietary XML/signature implementations can be plugged in. See Chapter 9.

TABLE 2–1 Public Interfaces *(Continued)*

Package Name	Description
`com.sun.identity.saml.assertion`	Provides classes to manage assertions and profiles. See Chapter 9.
`com.sun.identity.saml.common`	Provides classes that are common to all SAML elements. See Chapter 9.
`com.sun.identity.saml.plugins`	Provides SPIs to integrate SAML into custom services. See Chapter 9.
`com.sun.identity.saml.protocol`	Provides classes that parse the XML messages used to exchange assertions and information. See Chapter 9.
`com.sun.identity.saml.xmlsig`	Provides an SPI in which proprietary XML/signature implementations can be plugged in. See Chapter 9.
`com.sun.liberty`	Provides interfaces common to the Access Manager Federation Management module. See Chapter 3.

For more information, see Chapter 10. For detailed API documentation, including classes, methods and their syntax and parameters, see the Java API Reference in */AccessManager-base*`/SUNWam/docs` or on docs.sun.com.

SAML Service

Access Manager uses SAML as the means for exchanging security information. SAML uses an eXtensible Markup Language (XML) framework to achieve interoperability between vendor platforms that provide SAML assertions.

In anticipation of the next release of Access Manager and support of SAML 2.0, SAML attributes have been moved under the Federation tab although it's usage is independent of the functionality discussed in this guide. The Liberty-based features in Access Manager use SAML but that usage is not configurable. For more information on the independent SAML Service, see Chapter 9.

Liberty-Based Samples

Access Manager has included sample code and files that can be used to understand the implementation of the Liberty Alliance Project specifications. For information about the specifics of these samples, see the individual chapters or Appendix A.

PART II

Federation Management

◆ ◆ ◆ CHAPTER 3

Federation

Sun Java™ System Access Manager provides an interface for creating, modifying, and deleting authentication domains, service providers, and identity providers. This chapter explains how to use the Federation component to configure a Liberty-based provider federation.

This chapter covers the following topics:

Features of Federation

Access Manager provides a web interface to the Liberty Identity Federation Framework (Liberty ID-FF) which is accessible through the Federation tab in the Access Manager Console. The Federation component includes the features and functions described in the following sections.

Note – For more information about the Liberty ID-FF functions, see the *Liberty ID-FF Protocols and Schema Specifications*.

Identity Federation and Single Sign-On

Let's assume that a principal has separate user accounts with both a service provider and an identity provider in the same authentication domain. In order to gain access to these individual accounts, the

principal authenticates with each provider. After the principal has authenticated with the service provider though, they can be given the option to federate the service provider account with an identity provider account. Consenting to the federation of these two accounts links them for the purpose of single sign-on.

Providers differentiate between federated users by defining a unique *handle* for each account. (They are not required to use the principal's actual provider account identifier.) Providers can also choose to create multiple handles for a particular principal. However, identity providers must create one handle per user for each service provider that has multiple web sites so that the handle can be resolved across all of them.

Note – Because both the identity provider and service provider in a federation need to remember the principal's handle, they create entries that note the handle in their respective user repositories. In some scenarios, only the identity provider's handle is conveyed to a service provider. For example, if a service provider does not maintain its own user repository, the identity provider's handle is used.

Access Manager can accommodate the following functions:

- Providers of either type give the principal notice upon identity federation or identity defederation.
- Providers of either type notify each other regarding a principal's defederation.
- Identity providers notify the appropriate service providers regarding a principal's account termination.
- Providers of either type give the principal a list of their federated identities.
- Users can terminate federations or defederate identities.

Auto-Federation

Auto federation will automatically federate a user's disparate provider accounts based on a common attribute. During single sign-on, if it is deemed a user at provider A and a user at provider B have the same value for the defined common attribute (for example, an email address), the two accounts will be federated without consent or interaction from the principal. For more information, see "Auto-Federation" on page 96.

Bulk Federation

Federating one user's service provider account with their identity provider account generally requires the principal to visit both providers and link them. In situations when an enterprise is both a service provider and an identity provider, the organization should have the ability to federate user accounts behind the scenes. Access Manager provides a script for federating user accounts in bulk. The script allows the administrator to federate many (or all) of a principal's provider accounts based on metadata passed to the script. Bulk federation is useful when adding a new service provider to an enterprise so you can federate a group of existing employees to the new service. For more information, see "Bulk Federation" on page 97.

Authentication and Authentication Context

Single sign-on is the means by which a provider of either type can convey to another provider that the principal has been authenticated. Identity providers use local (to the identity provider) session information that is mapped to a user agent as the basis for issuing SAML authentication assertions to service providers. Thus, when the principal uses a user agent to interact with a service provider, the service provider requests authentication information from the identity provider based on the user agent's session information. If this information indicates that the user agent's session is presently active, the identity provider will return a positive authentication response to the service provider.

Access Manager accommodates these authentication actions:

- Supports a range of authentication methods (for example, password or certificate-based SSL).
- Allows providers to exchange the following minimum set of authentication information with regard to a principal: authentication status (active or not), instant (time authenticated), method, and pseudonym (temporary or persistent).
- Allows an identity provider, at the discretion of the service provider, to authenticate a principal by using an identity provider other than itself (proxy) and relay this information back to the service provider.

SAML is used for provider interaction during authentication but not all SAML assertions are equal. Different authorities issue SAML assertions of different quality. Therefore, the Liberty Alliance Project defines how the consumer of a SAML assertion can determine the amount of assurance to place in the assertion. This is referred to as the *authentication context*, information added to the SAML assertion that gives the assertion consumer information they need to make an informed entitlement decision. For example, a principal uses a simple identifier and a self-chosen password to authenticate to an identity provider. The identity provider sends an assertion that states the principal has been authenticated to a service provider. By sending the authentication context, the service provider can place the appropriate level of assurance on the associated authentication assertion. For example, if the service provider were a bank, they might require stronger authentication than that which has been used and send a response to the identity provider with a request to authenticate the user again. The authentication context information might include:

- The initial user identification mechanism (for example, face-to-face, online, or shared secret).
- The mechanisms for minimizing compromise of credentials (for example, private key in hardware, credential renewal frequency, or client-side key generation).
- The mechanisms for storing and protecting credentials (for example, Smartcard, or password rules).
- The authentication mechanisms (for example, password or Smartcard with PIN).

An XML schema has been defined by which the authority can assert the context of the SAML assertions it issues. The Liberty Alliance Project specifications have also defined Authentication Context *classes* against which an identity provider can claim conformance. The Liberty-defined authentication contexts are listed and described in the following table.

TABLE 3–1 Authentication Context Classes

Class	Description
MobileContract	Identified when a mobile principal has an identity for which the identity provider has vouched.
MobileDigitalID	Identified by detailed and verified registration procedures, a user's consent to sign and authorize transactions, and DigitalID-based authentication.
MobileUnregistered	Identified when the real identity of a mobile principal has not been strongly verified.
Password	Identified when a principal authenticates to an identity provider by using a password over an unprotected HTTP session.
Password-ProtectedTransport	Identified when a principal authenticates to an identity provider by using a password over an SSL-protected session.
Previous-Session	Identified when an identity provider must authenticate a principal for a current authentication event and the principal has previously authenticated to the identity provider. This affirms to the service provider a time lapse from the principal's current resource access request. **Note –** The context for the previously authenticated session is not included in this class because the user has not authenticated during this session. Thus, the mechanism that the user employed to authenticate in a previous session should not be used as part of a decision on whether to now allow access to a resource.
Smartcard	Identified when a principal uses a smart card to authenticate to an identity provider.
Smartcard-PKI	Identified when a principal uses a smart card with an enclosed private key and a PIN to authenticate to an identity provider.
Software-PKI	Identified when a principal uses an X.509 certificate stored in software to authenticate to the identity provider over an SSL-protected session.
Time-Sync-Token	Identified when a principal authenticates through a time synchronization token.

Note – For more information on authentication context, see the *Liberty ID-FF Authentication Context Specification*.

Identifiers and Name Registration

Access Manager supports name identifiers that are unique across all providers in an authentication domain. This identifier can be used to obtain information for or about the principal (with consent) without requiring the user to consent to a long-term relationship with the service provider. During federation, the identity provider generates an opaque value that serves as the initial name identifier that both the service provider and the identity provider use to refer to the principal when communicating with each other.

After federation though, the identity provider or the service provider may register a different opaque value. The reasons for doing this would be implementation-specific. If a service provider registers a different opaque value for the principal, the identity provider must use the new identifier when communicating with the service provider about the principal.

Note – The initial name identifier defined by the identity provider is always used to refer to the principal unless a new name identifier is registered.

Global Logout

A principal may establish authenticated sessions with both an identity provider and individual service providers, based on authentication assertions supplied by the identity provider. When the principal logs out of a service provider session, the service provider sends a logout message to the identity provider that provided the authentication for that session. When this happen, or the principal manually logs out of a session at an identity provider, the identity provider sends a logout message to each service provider to which it provided authentication assertions under the relevant session. The one exception is the service provider that sent the logout request to the identity provider.

Dynamic Identity Provider Proxying

An identity provider can choose to proxy an authentication request to an identity provider in another authentication domain if it knows that the principal has been authenticated with this identity provider. The proxy behavior is defined by the local policy of the proxying identity provider. However, a service provider can override this behavior and choose not to proxy. This function can be implemented as a form of authentication when, for instance, a roaming mobile user accesses a service provider that is not part of the mobile home network. For more information see "Dynamic Identity Provider Proxying" on page 97.

Process of Federation

The process of federation begins with authentication. A standard installation of Access Manager provides two options for user authentication: the proprietary Authentication Service and the Liberty-based Federation component.

With the proprietary option, users attempting to access a resource protected by Access Manager are redirected to the Authentication Service via an Access Manager login page. After the users provide credentials, the Authentication Service allows or denies access to the resource based on the outcome.

Note – For more information about the proprietary Authentication Service, see the *Sun Java System Access Manager 7 2005Q4 Administration Guide*.

The second option for user authentication is Liberty-based federation. When a principal attempts to access a web site that belongs to the trusted member provider of a configured authentication domain, the process of user authentication begins with the search for a valid Access Manager session token from the proprietary Authentication Service.

- If no session token is found, the principal is redirected to a location defined by the pre-login URL to establish a valid session. See "Pre-login Process" on page 66 for details.
- If a session token is found, the principal is granted (or denied) access to the requested page. Assuming access is granted, the requested page contains a link so the principal can federate the Access Manager identity with the identity local to the requested site. If the principal clicks this link, federation begins. See "Federation and Single Sign-On" on page 66 for details.

The following figure illustrates these divergent paths.

Note – The process shown in the figure below is the default process when no application has been deployed. When an application is deployed and using Access Manager, the process will change based on the application's query parameters and preferences. For more information, see "The Pre-login URL" on page 99.

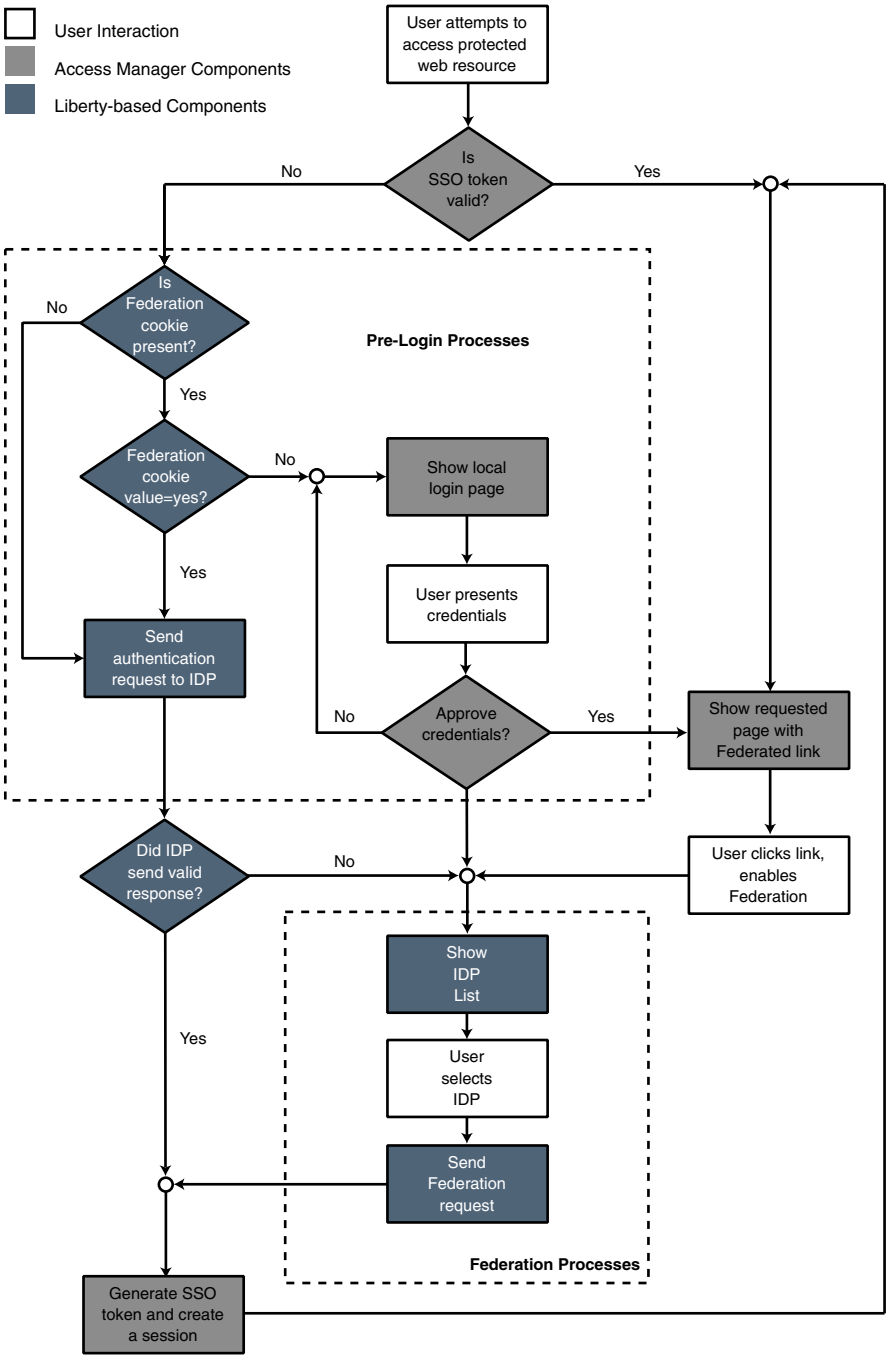

FIGURE 3–1 Default Process of Federation

Pre-login Process

The pre-login process establishes a valid Access Manager session. When a principal attempts to access a service provider site and no Access Manager session token is found, Access Manager searches for a federation cookie. A *federation cookie* is implemented by Access Manager and is called `fedCookie`. It can have a value of either `yes` or `no`, based on the principal's federation status.

Note – A federation cookie is *not* defined in the Liberty Alliance Project specifications.

At this point, the pre-login process may take one of the following paths:

- If a federation cookie is found and its value is `no`, an Access Manager login page is displayed and the principal submits credentials to the proprietary Authentication Service. When authenticated by Access Manager, the principal is redirected to the requested page, which might contain a link to allow for identity federation. If the principal clicks this link, federation begins. See "Federation and Single Sign-On" on page 66 for details.
- If a federation cookie is found and its value is `yes`, the principal has already federated an identity but has not been authenticated by an identity provider within the authentication domain for this Access Manager session. Authentication to Access Manager is achieved on the back end by sending a request to the principal's identity provider. After authentication, the principal is directed back to the requested page.
- If no federation cookie is found, a *passive* authentication request (one that does not allow identity provider interaction with the principal) is sent to the principal's identity provider. If an affirmative authentication is received back from the identity provider, the principal is directed to the Access Manager Authentication Service, where a session token is granted. The principal is then redirected to the requested page. If the response from the identity provider is negative (for example, if the session has timed out), the principal is sent to a common login page to complete either a local login or Liberty-based federation. See "Federation and Single Sign-On" on page 66 for details.

Note – This pre-login process is the default behavior of Access Manager. This process might change based on parameters passed to Access Manager from the participating application. For more details, see the section on "The Pre-login URL" on page 99.

Federation and Single Sign-On

When a principal logs in to access a protected resource or service, Access Manager sends a request to the appropriate identity provider for authentication confirmation. If the identity provider sends a positive response, the principal gains access to all provider sites within the authentication domain. If the identity provider sends a negative response, the principal is directed to authenticate again using the Liberty-based federation process.

In the Liberty-based federation process, a principal selects an identity provider and sends credentials for authentication. After authentication is complete and access is granted, the principal is issued a

session token from the Access Manager Authentication Service and redirected to the requested page. As long as the session token remains valid, the principal can access other service providers in the authentication domain without having to authenticate again.

Note – Common Domain Services are used by a service provider to determine the identity provider used by a principal in an authentication domain that contains multiple identity providers. See Chapter 4 for details.

Federation Graphical User Interface

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

The JSP are located in `/`*AccessManager-base*`/SUNWam/web-src/services/config/federation/default`. The files in this directory provide a default interface to the Federation component. To customize the pages for a specific organization, this default directory can be copied and renamed to reflect the name of the organization (or any value). This directory would then be placed at the same level as the default directory, and the files within this directory would be modified as needed. The following table lists the JSP including details on what each page is used for and the invoked APIs that cannot be modified. For more information about modifying these pages to customize the console, see the *Sun Java System Access Manager 7 2005Q4 Developer's Guide*.

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

JSP Name and Implemented APIs	Purpose
■ `ListOfCOTs.jsp` Invoked API is: ■ `LibertyManager.getListOfCOTs(providerID)`	Displays a list of circles of trust. When a user is authenticated by an identity provider and the service provider belongs to more than one circle of trust, the user is shown this JSP and is prompted to select an authentication domain as their preferred domain. In the case that the provider belongs to only one domain, this page will not be displayed. The list is obtained by using the `getListOfCOTs(providerID)` method.
■ `LogoutDone.jsp` Invoked API is: ■ `LibertyManager.isLogoutSuccess(request)`	Displays the status of the local logout operation.
■ `NameRegistration.jsp` Invoked APIs are: ■ `LibertyManager.getUser(request)` ■ `LibertyManager.getRegisteredProviders(userDN)`	Displays when the Name Registration link is clicked on a provider page. When a federated user chooses to register a new Name Identifier from a service provider to an identity provider, this JSP is displayed.
■ `NameRegistrationDone.jsp` Invoked APIs are: ■ `LibertyManager.isNameRegistrationSuccess(request)` ■ `LibertyManager.isNameRegistrationCanceled(request)`	Displays the status of `NameRegistration.jsp`. When finished, this page is displayed.
■ `Termination.jsp` Invoked APIs are: ■ `LibertyManager.getUser(request)` ■ `LibertyManager.getFederatedProviders(userDN)`	Displays when a user clicks a defederate link on a provider page. Contains a drop-down of all providers to which the user has federated and from which the user can choose to defederate. The list is constructed by using the `getFederatedProviders(userName)` method, which returns all active providers to which the user is already federated.
■ `TerminationDone.jsp` Invoked APIs are: ■ `LibertyManager.isTerminationSuccess(request)` ■ `LibertyManager.isTerminationCanceled(request)`	Displays the status of federation termination (success or cancelled). Status is checked using the `isTerminationCancelled(request)` method.

Entities and Authentication Domains

The Federation component in the Access Manager Console provides an interface for configuring, modifying, and deleting authentication domains, and identity and service providers. To create and populate an authentication domain, you first create an *entity* to hold the metadata for each provider that will become a member of the authentication domain. Then, you configure and save the authentication domain. Finally, to add an entity to the authentication domain, you edit the entity's properties. The following sections contain more information:

- "Entities" on page 70
- "Authentication Domains" on page 93

Note – In a federation setup, all service providers and identity providers must share a synchronized clock. You can implement the synchronization by pointing to an external clock source or by ensuring that, in case of delays in receiving responses, the responses are captured without fail through adjustments of the timeouts.

Entities

In Access Manager an *entity* can contain configuration information for an individual identity provider, an individual service provider, or one of each. An entity can also contain configuration information for an *affiliation*, a group of providers of either type. Both provider and affiliation entities can be configured using the Access Manager Console.

Note – For general information about entities, see the *Liberty Metadata Description and Discovery Specification*.

Provider Entity A *provider entity* holds the metadata for individual providers of either type. All identity providers and service providers must first be configured within a provider entity. After they are configured in an entity, they can be associated with an authentication domain, or chosen to be included in an *affiliate entity*. Using the descriptor attributes, one individual identity provider, one individual service provider, or one of each can be defined within a provider entity.

Affiliate Entity An *affiliate entity* holds the metadata that defines a group of one or more providers that was formed without regard to the boundaries of an authentication domain. This *affiliation* (referenced by an affiliationID) is formed and maintained by an *affiliation owner* (referenced by the providerID of the entity that defined it) who chooses the trusted providers from already configured provider entities. Members of the affiliation may invoke services either as a member of the affiliation (using the affiliationID), or individually (using their providerID). For example, when a service provider issues an authentication request on behalf of an affiliation, the AffiliationID will be used to achieve single sign-on and the identity provider will resolve federations based on the same AffiliationID. The affiliate entity itself does not contain the

configuration information for any providers, only the configuration information for the entity.

Note – The name identifier (a single persistent randomized string) is used to achieve single sign-on between an identity provider and a group of service providers acting as a single affiliation. If there are several service providers and identity providers in the same circle of trust, use an affiliate entity to avoid having to generate different name identifiers for commonly shared services.

Creating an entity is a two-step process. First, you create a provider or affiliate entity. Then, you populate the entity with remote or hosted provider information (either service or identity) or affiliation information. This process is described in the following sections:

- "Creating Entities" on page 71
- "Configuring Provider Entities" on page 72
- "Configuring Affiliate Entities" on page 90
- "Deleting Entities" on page 93

Creating Entities

This section describes the process for creating a provider entity or an affiliate entity.

▼ To Create a Provider Entity or an Affiliate Entity

An entity can be created but it will not be available for assignment to an authentication domain until it has been populated with provider(s). Once created and configured, the entity (and thus the providers) can be added to an authentication domain.

1 In the Access Manager Console, select the Federation tab.

2 Under Federation, select the Entities tab.

3 Select New.

The new entity attributes are displayed.

4 Type a value for the Entity Name.

This field specifies the Uniform Resource Identifier (URI) of the entity and must be unique. For example, `http://shivalik.sun.com` or `http://provider2.com:875`.

5 (Optional) Enter a description of the entity in the Description field.

6 **Select one of the following options to define the entity's type.**

- **Select Provider and click OK.**

  The new entity is now displayed as a provider in the list of configured Entities. To configure the entity, see "To Configure a Provider Entity" on page 72.

- **Select Affiliate, type a value for both Affiliate Name and Affiliate Owner, and click OK.**

  The Affiliate Name (also referred to as the affiliation ID) specifies a URI defined by the Affiliate Owner that uniquely represents the affiliate entity, for example, `http://shivalik.sun.com` or `http://provider2.com:875`. The Affiliate Owner is the provider ID of the service provider (defined in a provider entity) that is forming the affiliation. After entering these values and clicking OK, the new entity is displayed as an affiliate in the list of configured Entities. To configure the entity, see "To Configure an Affiliate Entity" on page 90.

Note – Defining a service provider as the Affiliate Owner does not automatically include it as a member of the affiliate. If an owner is also a member, the provider ID must be defined in both attributes.

Configuring Provider Entities

After you create a provider entity, you populate it with remote or hosted provider information (either service or identity). This section contains the following procedures:

▼ To Configure a Provider Entity

When you configure a provider entity, you are populating it with remote or hosted provider information (either service or identity). You might also be configuring values for attributes that were not available when the entity was initially created.

1 **In the Access Manager Console, select the Federation tab.**

2 **Under Federation, select the Entities tab.**

3 **Select the provider entity that you want to configure.**

Ensure that you select an entity marked as type Provider.

4 **Define values for the General, Identity Provider or Service Provider attributes by choosing from the View menu:**

- **To define values for General attributes, see "To Configure General Attributes for a Provider Entity" on page 73.**
- **To define values for Identity Provider attributes, see "To Configure Hosted or Remote Identity Provider Attributes for a Provider Entity" on page 75.**
- **To define values for Service Provider attributes, see "To Configure Hosted or Remote Service Provider Attributes for a Provider Entity" on page 82.**

▼ To Configure General Attributes for a Provider Entity

Before performing this procedure, you must have completed the steps in "To Configure a Provider Entity" on page 72.

1 **Choose General from the View menu, and provide information for the Entity Common Attributes.**

Entity Common Attributes contain values that define the entity itself.

Entity Name
: The static value of this attribute is the name that you provided when creating the entity.

Type
: The static value of this attribute is Provider.

Description
: The value of this optional attribute is the description that you provided when creating the entity. You can modify the description.

Valid Until
: Type the expiration date for the entity metadata. Use Coordinated Universal Time (UTC) in the format *yyyy-mm-dd*T*hh:mm:ss.S*Z. For example, `2004-12-31T14:30:00.0Z`.

Cache Duration
: Type the maximum amount of time that the entity metadata can be cached. Use the format P*n*Y*n*M*n*DT*n*H*n*M*n*S, where *n* is an integer variable. For example, `P1Y2M4DT9H8M20S` defines the cache duration as 1 year, 2 months, 4 days, 9 hours, 8 minutes, and 20 seconds.

2 **Provide information for the Entity Contact Person Profile attributes.**

Entity Contact Person Profile attributes contain values that define the administrator of the entity.

First Name
: Type the given name of the entity's contact person.

Last Name
: Type the surname of the entity's contact person.

Type
: Choose the type of contact from the drop-down menu:

- Administrative
- Billing
- Technical
- Other

Company
Type the name of the company that employs this person.

Liberty Principal ID
Type a URI that points to an online instance of the contact person's personal information profile.

Emails
Type one or more email addresses for the contact person.

Telephone Numbers
Type one or more telephone numbers for the contact person.

3 (Optional) Provide information for the Organization Profiles.

The Organization Profiles attributes contain values that define the organizational name of the entity.

Names
Type the complete legal name of the entity's organization. Use the format *locale*|*organization-name*. For example, `en|`*organization-name*`.com`.

Note – If the Names attribute contains a value, it is required to add values to the Display Names and URL attributes.

Display Names
Type a name that is suitable for display. Use the format *locale*|*organization-display-name*. For example, `en|`*organization-display-name*`.com`.

URL
Type a URL that can be used to direct a principal to additional information on the entity's organization. Use the format *locale*|*organization-URL*. For example, `en|http://www.`*organization-name*`.com`.

4 Click Save to complete the configuration, or define values for Identity Provider or Service Provider attributes by choosing from the View menu:

- **To define values for Identity Provider attributes, see "To Configure Hosted or Remote Identity Provider Attributes for a Provider Entity" on page 75.**
- **To define values for Service Provider attributes, see "To Configure Hosted or Remote Service Provider Attributes for a Provider Entity" on page 82.**

▼ To Configure Hosted or Remote Identity Provider Attributes for a Provider Entity

Before performing this procedure, you must have completed the steps in "To Configure a Provider Entity" on page 72.

1 Choose Identity Provider from the View menu.

2 Select the type of provider that you are configuring:

- New Hosted Provider
- New Remote Provider

A *hosted provider* is installed on the same server as Access Manager. A *remote provider* is not installed on the same server as Access Manager.

3 Provide information for the Common Attributes.

Common Attributes contain values that generally define the identity provider.

Provider Type
: The static value of this attribute is the type of provider being configured: hosted or remote. This attribute is visible only after saving your configuration.

Description
: The value of this attribute is a description of the identity provider.

Valid Until
: Type the expiration date for the provider metadata. Use Coordinated Universal Time (UTC) and the format *yyyy-mm-dd*T*hh:mm:ss.SZ*, for example, `2004-12-31T14:30:00.0Z`.

Cache Duration
: Type the maximum amount of time that the provider metadata can be cached. Use the format P*n*Y*n*M*n*DT*n*H*n*M*n*S, where *n* is an integer. For example, `P1Y2M4DT9H8M20S` defines the cache duration as 1 year, 2 months, 4 days, 9 hours, 8 minutes, and 20 seconds.

Protocol Support Enumeration
: Choose the Liberty ID-FF release that is supported by this provider.

  - `urn:liberty:iff:2003-08` refers to the Liberty Identity Federation Framework Version 1.2.
  - `urn:liberty:iff:2002-12` refers to the Liberty Identity Federation Framework Version 1.1.

Server Name Identifier Mapping Binding
: Name identifier mapping allows a service provider to obtain a name identifier for a principal that has federated in the namespace of a different service provider. Implementing this protocol allows the requesting service provider to communicate with the second service provider without an identity federation having been enabled. Type a URI that identifies the communication specifications.

Note – Currently, the Name Identifier Mapping profile only supports SOAP. If this attribute is used, its value must be `http://projectliberty.org/profiles/nim-sp-http`.

Additional Meta Locations
Type a URL that points to other relevant metadata concerning the provider.

Signing Key: Key Alias
Type the key alias that is used to sign requests and responses.

Encryption Key: Key Alias
Type the security certificate alias. Certificates are stored in a Java keystore file. Each specific certificate is mapped to an alias that is used to fetch the certificate.

Encryption Key: Key Size
Type the length for keys that are used by the web service consumer when interacting with another entity.

Note – If the encryption method is DESede, the key size must be 192. If the encryption method is AES, the key size must be 128, 192 or 256.

Encryption Key: Encryption Method
Choose the method of encryption:

- None
- AES
- DESede

Name Identifier Encryption
Select the check box to enable encryption of the name identifier.

4 Provide information for the Communication URLs.

Communication URLs attributes contain locations for redirects and sending requests.

SOAP Endpoint
Type a URI to the identity provider's SOAP message receiver. This value communicates the location of the SOAP receiver in non browser communications.

Single Sign-On Service URL
Type a URL to which service providers can send single sign-on and federation requests.

Single Logout Service
Type a URL to which service providers can send logout requests. Single logout synchronizes the logout functionality across all sessions authenticated by the identity provider.

Single Logout Return
Type a URL to which the identity provider will redirect the principal after completing a logout.

Federation Termination Service
Type a URL to which a service provider will send federation termination requests.

Federation Termination Return
: Type a URL to which the identity provider will redirect the principal after completing federation termination.

Name Registration Service
: Type a URL to which a service provider will send requests to specify a new name identifier to be used when communicating with the identity provider about a principal. This service can only be used after a federation session is established.

Name Registration Return
: Type a URL to which the identity provider will redirect the principal after HTTP name registration has been completed.

5 Provide information for the Communication Profiles.

Communication Profiles attributes define the transmission methods used by the identity provider.

Federation Termination
: Select a profile to notify other providers of a principal's federation termination:

- HTTP Redirect
- SOAP

Single Logout
: Select a profile to notify other providers of a principal's logout:

- HTTP Redirect
- HTTP Get
- SOAP

Name Registration
: Select a profile to notify other providers of a principal's name registration:

- HTTP Redirect
- SOAP

Single Sign-on/Federation
: Select a profile for sending authentication requests:

- Browser Post (specifies a browser-based HTTP POST protocol)
- Browser Artifact (specifies a non-browser SOAP-based protocol)
- LECP (specifies a Liberty-enabled Client Proxy)

Note – Access Manager can handle requests that come from a Liberty-enabled client proxy profile, but it requires additional configuration that is beyond the scope of this manual.

---

6 Select any of the available authentication domains to assign to the provider.

A provider can belong to one or more authentication domains. However, a provider without a specified authentication domain can not participate in Liberty-based communications. If no authentication domains have been created, you can define this attribute later.

Note – To continue configuring a remote identity provider, skip to step 11.

7 (Hosted Identity Provider Only) Provide mappings for the Authentication Context classes.

This attribute maps the Liberty-defined authentication context classes to authentication methods available at the identity provider.

Supported
Select the check box next to the authentication context class if the identity provider supports it.

Context Reference
The Liberty-defined authentication context classes are:

- Password
- Mobile Digital ID
- Smartcard
- Smartcard-PKI
- MobileUnregistered
- Software-PKI
- Previous-Session
- Mobile Contract
- Time-Sync-Token
- Password-ProtectedTransport

Key
Choose the Access Manager authentication type to which the context is mapped.

Value
Type the Access Manager authentication option.

Priority
Choose a priority level for cases where there are multiple contexts.

8 (Hosted Identity Provider Only) Select any of the available provider entities to assign as a Trusted Provider and click Add.

This attribute tallies providers that the identity provider trusts. It is visible after the provider configuration has been saved.

9 (Hosted Identity Provider Only) Provide information for the Access Manager Configuration attributes.

Access Manager Configuration attributes define general information regarding the instance of Access Manager being used as an identity provider.

Provider URL
Type the URL of the local identity provider.

Provider Alias
Type an alias name for the local identity provider.

Authentication Type
Select the provider that should be used for authentication requests from a provider hosted locally:

- *Remote* specifies that the provider hosted locally would contact a remote identity provider upon receiving an authentication request.
- *Local* specifies that the provider hosted locally should contact a local identity provider upon receiving an authentication request (essentially, itself).

Default Authentication Context
Select the authentication context class (method of authentication) to use if the identity provider does not receive this information as part of a service provider request. This value also specifies the authentication context used by the service provider when an unknown user tries to access a protected resource. The options are as follows:

- Password
- Mobile Digital ID
- Smartcard
- Smartcard-PKI
- MobileUnregistered
- Software-PKI
- Previous-Session
- Mobile Contract
- Time-Sync-Token
- Password-ProtectedTransport

Identity Provider Forced Authentication
Select the check box to indicate that the identity provider must reauthenticate (even during a live session) when an authentication request is received. This attribute is enabled by default.

Request Identity Provider to be Passive
Select the check box to specify that the identity provider must not interact with the principal and must interact with the user.

Realm
Type a value that points to the realm in which this provider is configured. For example, `/sp`.

Liberty Version URI
Type the URI of the version of the Liberty Alliance Project specification being used. The default value is `http://projectliberty.org/specs/v1`.

Name Identifier Implementation
This field defines the class used by a service provider to participate in name registration. Name registration is a profile by which service providers specify a principal's name identifier that an identity provider will use when communicating with the service provider. The value is `com.sun.identity.federation.services.util.FSNameIdentifierImpl`.

Home Page URL
Type the URL of the home page of the identity provider.

Single Sign-on Failure Redirect URL
: Type the URL to which a principal will be redirected if single sign-on has failed.

Assertion Issuer
: Type the name of the host that issues the assertion. This value might be the load balancer's host name if Access Manager is behind one.

Generate Discovery Bootstrapping Resource Offering
: Select the check box if you want a Discovery Service Resource Offering to be generated during the Liberty-based single sign-on process for bootstrapping purposes.

Auto Federation
: Select the check box to enable auto-federation.

Auto Federation Common Attribute Name
: When creating an Auto Federation Attribute Statement, the value of this attribute will be used. The statement will contain the `AutoFedAttribute` element and this common attribute as its value.

Attribute Statement Plugin
: Specify a pluggable class used for adding attribute statements to an assertion that is generated during the Liberty-based single sign-on process.

10 (Hosted Identity Provider Only) Provide information for the SAML Attributes.

SAML Attributes define general information regarding SAML assertions that are sent by the identity provider.

Assertion Interval
: Type the interval of time (in seconds) that an assertion issued by the identity provider will remain valid. A principal will remain authenticated until the assertion interval expires.

Cleanup Interval
: Type the interval of time (in seconds) before assertions stored in the identity provider will be cleared.

Artifact Timeout
: Type the interval of time (in seconds) to specify the timeout for assertion artifacts.

Assertion Limit
: Type a number to define how many assertions an identity provider can issue, or how many assertions that can be stored.

Note – To continue configuring a hosted identity provider, skip to step 12.

---

11 (Remote Identity Provider Only) Provide information for the Proxy Authentication Configuration attributes.

Proxy Authentication Configuration attributes define values for dynamic identity provider proxying.

Enable Proxy Authentication
: Select the check box to enable proxy authentication for a service provider.

Proxy Identity Providers List
: Add a list of identity providers that can be used for proxy authentication. The value is a URI defined as the provider's identifier.

Maximum Number of Proxies
: Enter the maximum number of identity providers that can be used for proxy authentication.

Use Introduction Cookie for Proxying
: Select the check box if you want introductions to be used to find the proxying identity provider.

12 Provide information for the Organization Profiles.

The optional Organization Profiles attributes contain values that define the organizational name of the entity.

Names
: Type the complete legal name of the organization. Use the format *locale*|*organization-name*, for example, `en|`*organization-name*`.com`.

Note – If the Names attribute contains a value, it is required to add values to the Display Names and URL attributes also.

Display Names
: Type a name that is suitable for display to a principal. The value is defined in the format *locale*|*organization-display-name*, for example, `en|`*organization-display-name*`.com`.

URL
: Type a URL that can be used to direct a principal to additional information on the entity. Use the format *locale*|*organization-URL*, for example, `en|http://www.`*organization-name*`.com`.

13 Click New Contact Person to create a contact person for the provider.

The Contact Person attributes contain information regarding a human contact for the identity provider.

First Name
: Type the given name of the identity provider's contact person.

Last Name
: Type the surname of the identity provider's contact person.

Type
: Choose the contact's role from the drop-down menu:

  - Administrative
  - Billing
  - Technical
  - Other

Company
: Type the name of the company that employs the contact person.

Liberty Principal Identifier
: Type the name identifier that points to an online instance of the contact person's personal information profile.

Emails
: Type one or more email addresses for the contact person.

Telephone Numbers
: Type one or more telephone numbers for the contact person.

14 Click Create to create the contact person.

15 Click Save to complete the configuration, or define values for General or Service Provider attributes by choosing from the View menu:

- **To define values for General attributes, see "To Configure General Attributes for a Provider Entity" on page 73.**
- **To define values for Service Provider attributes, see "To Configure Hosted or Remote Service Provider Attributes for a Provider Entity" on page 82.**

▼ To Configure Hosted or Remote Service Provider Attributes for a Provider Entity

Before performing this procedure, you must have completed the steps in "To Configure a Provider Entity" on page 72.

1 Choose Service Provider from the View menu.

2 Select the type of provider that you are configuring:

- New Hosted Provider
- New Remote Provider

A *hosted provider* is installed on the same server as Access Manager. A *remote provider* is not installed on the same server as Access Manager.

3 Provide information for the Common Attributes.

Common Attributes contain values that generally define the service provider.

Provider Type
: The static value of this attribute is the type of provider being configured: hosted or remote. This attribute is visible only after saving your configuration.

Description
: The value of this attribute is a description of the service provider.

Valid Until
Type the expiration date for the provider metadata. Use Coordinated Universal Time (UTC) and the format *yyyy-mm-dd*T*hh:mm:ss.S*Z, for example, `2004-12-31T14:30:00.0Z`.

Cache Duration
Type the maximum amount of time that the provider metadata can be cached. Use the format P*n*Y*n*M*n*DT*n*H*n*M*n*S, where *n* is an integer. For example, `P1Y2M4DT9H8M20S` defines the cache duration as 1 year, 2 months, 4 days, 9 hours, 8 minutes, and 20 seconds.

Protocol Support Enumeration
Select the Liberty ID-FF release that is supported by this provider.

- `urn:liberty:iff:2003-08` refers to the Liberty Identity Federation Framework Version 1.2.
- `urn:liberty:iff:2002-12` refers to the Liberty Identity Federation Framework Version 1.1.

Server Name Identifier Mapping Binding
Name identifier mapping allows a service provider to obtain a name identifier for a principal that has federated in the namespace of a different service provider. Implementing this protocol allows the requesting service provider to communicate with the second service provider without an identity federation having been enabled. The value of this attribute is a URI that identifies the communication specifications.

Note – Currently, the Name Identifier Mapping profile only supports SOAP. If this attribute is used, its value must be `http://projectliberty.org/profiles/nim-sp-http`.

Additional Meta Locations
Type a URL that points to other relevant metadata concerning the provider.

Signing Key: Key Alias
Type the key alias that is used to sign requests and responses.

Encryption Key: Key Alias
Type the security certificate alias. Certificates are stored in a Java keystore file. Each specific certificate is mapped to an alias that is used to fetch the certificate.

Encryption Key: Key Size
Type the length for keys that are used by the web service consumer when interacting with another entity.

Encryption Key: Encryption Method
Select the method of encryption:

- None
- AES
- DESede

Name Identifier Encryption
Select the check box to enable encryption of the name identifier.

4 Provide information for the Communication URLs.

Communication URLs attributes contain locations for redirects and sending requests.

SOAP Endpoint
Type a URI to the service provider's SOAP message receiver. This value communicates the location of the SOAP receiver in non browser communications.

Single Logout Service
Type a URL to which identity providers can send logout requests.

Single Logout Return
Type a URL to which the service provider will redirect the principal after completing a logout.

Federation Termination Service
Type a URL to which identity providers will send federation termination requests.

Federation Termination Return
Type a URL to which the service provider will redirect the principal after completing federation termination.

Name Registration Service
Type a URL that will be used when communicating with the identity provider to specify a new name identifier for the principal. (Registration can occur only after a federation session is established.)

Name Registration Return
Type a URL to which the service provider will redirect the principal after HTTP name registration has been completed.

5 Provide information for the Communication Profiles.

Communication Profiles attributes define the transmission methods used by the service provider.

Federation Termination
Select a profile to notify other providers of a principal's federation termination:

- HTTP Redirect
- SOAP

Single Logout
Select a profile to notify other providers of a principal's logout:

- HTTP Redirect
- HTTP Get
- SOAP

Name Registration
Select a profile to notify other providers of a principal's name registration:

- HTTP Redirect
- SOAP

Single Sign-on/Federation
Select a profile for sending authentication requests:

- Browser Post (specifies a browser-based HTTP POST protocol)
- Browser Artifact (specifies a non-browser SOAP-based protocol)

- LECP (specifies a Liberty-enabled Client Proxy)

Note – Access Manager can handle requests that come from a Liberty-enabled client proxy profile, but it requires additional configuration that is beyond the scope of this manual.

6 Select any of the available authentication domains to assign to the provider.

A provider can belong to one or more authentication domains. However, a provider without a specified authentication domain cannot participate in Liberty-based communications. If no authentication domains have been created, you can define this attribute later.

Note – To continue configuring a remote service provider, skip to step 9.

7 (Hosted Service Provider Only) Provide a hierarchy for the Authentication Context classes.

This attribute corresponds to the authentication level defined for an Access Manager authentication module. It will redirect the principal to the authentication type with an authentication level equal to the number defined.

Context Reference

The Liberty-defined authentication context classes are:

- Password
- Mobile Digital ID
- Smartcard
- Smartcard-PKI
- MobileUnregistered
- Software-PKI
- Previous-Session
- Mobile Contract
- Time-Sync-Token
- Password-ProtectedTransport

Level

Type a level for each authentication context class. The number can be any positive number.

8 (Hosted Service Provider Only) Select any of the available provider entities to assign as a Trusted Provider and click Add.

This attribute tallies providers that the service provider trusts.

9 Provide information for the Service Provider attributes.

Service Provider attributes define general information regarding the service provider.

Assertion Consumer URL

Type the URL to the end point that defines where a provider will send SAML assertions.

Assertion Consumer Service URL ID

If the value of the Protocol Support Enumeration common attribute is `urn:liberty:iff:2003-08`, type the required ID.

Set Assertion Consumer Service URL as Default
Select the check box to use the Assertion Consumer Service URL as the default value when no identifier is provided in the request.

Sign Authentication Request
Select the check box to make the service provider always signs authentication requests.

Name Registration after Federation
Select the check box to enable the service provider to participate in name registration after a principal has been federated.

Name ID Policy
Select the option permitting requester influence over name identifier policy at the identity provider. The options are:

- *None* specifies that the identity provider will return the name identifier(s) for the principal corresponding to the federation that exists between the identity provider and the requesting service provider or affiliation group. If no such federation exists, an error will be returned.
- *One-time* specifies that the identity provider will issue a temporary, one-time-use identifier for the principal after federation.
- *Federation* specifies that the identity provider may start a new identity federation if one does not already exist for the principal.

Enable Affiliation Federation
Select the check box to enable affiliation federation.

Note – To continue configuring a remote identity provider, skip to step 11.

---

10 (Hosted Service Provider Only) Provide information for the Access Manager Configuration attributes.

Access Manager Configuration attributes define general information regarding the instance of Access Manager being used as a service provider.

Provider URL
Type the URL of the local service provider.

Provider Alias
Type an alias name for the local service provider.

Authentication Type
Select the provider that should be used for authentication requests from a provider hosted locally:

- *Remote* specifies that the provider hosted locally would contact a remote identity provider upon receiving an authentication request.
- *Local* specifies that the provider hosted locally should contact a local identity provider upon receiving an authentication request (essentially, itself).

Default Authentication Context
This attribute defines the service provider's default authentication context class (method of authentication). This method will always be called when the service provider sends an

authentication request. This value also specifies the authentication context used by the service provider when an unknown user tries to access a protected resource. The options are:

- Password
- Mobile Digital ID
- Smartcard
- Smartcard-PKI
- MobileUnregistered
- Software-PKI
- Previous-Session
- Mobile Contract
- Time-Sync-Token
- Password-ProtectedTransport

Identity Provider Forced Authentication
: Select the check box to indicate that the identity provider must reauthenticate (even during a live session) when an authentication request is received. This attribute is enabled by default.

Request Identity Provider to be Passive
: Select the check box to specify that the identity provider must not interact with the principal and must interact with the user.

Realm
: Type a value that points to the realm in which this provider is configured, for example, `/sp`.

Liberty Version URI
: Type the URI of the version of the Liberty specification being used. The default value is `http://projectliberty.org/specs/v1`.

Name Identifier Implementation
: This field defines the class used by a service provider to participate in name registration. Name registration is a profile by which service providers specify a principal's name identifier that an identity provider will use when communicating with the service provider. The value is `com.sun.identity.federation.services.util.FSNameIdentifierImpl`.

Home Page URL
: Type the URL of the home page of the service provider.

Single Sign-on Failure Redirect URL
: Type the URL to which a principal will be redirected if single sign-on has failed.

Auto Federation
: Select the check box to enable auto-federation.

Auto Federation Common Attribute Name
: When creating an Auto Federation Attribute Statement, the value of this attribute will be used. The statement will contain the `AutoFedAttribute` element and this common attribute as its value.

Attribute Statement Plugin
: Specify a pluggable class used for adding attribute statements to an assertion that is generated during the Liberty-based single sign-on process.

11 Provide information for the Proxy Authentication Configuration attributes.

Proxy Authentication Configuration attributes define values for dynamic identity provider proxying.

Enable Proxy Authentication
: Select the check box to enable proxy authentication for a service provider.

Proxy Identity Providers List
: Add a list of identity providers that can be used for proxy authentication. The value is a URI defined as the provider's identifier.

Maximum Number of Proxies
: Enter the maximum number of identity providers that can be used for proxy authentication.

Use Introduction Cookie for Proxying
: Select the check box if you want introductions to be used to find the proxying identity provider.

Note – To continue configuring a remote identity provider, skip to step 13.

---

12 (Hosted Service Provider Only) Provide information for the SAML Attributes.

SAML Attributes define general information regarding SAML assertions sent by the identity provider.

Assertion Interval
: Type the interval of time (in seconds) that an assertion issued by the identity provider will remain valid. A principal will remain authenticated until the assertion interval expires.

Cleanup Interval
: Type the interval of time (in seconds) before assertions stored in the identity provider will be cleared.

Artifact Timeout
: Type the interval of time (in seconds) to specify the timeout for assertion artifacts.

Assertion Limit
: Type a number to define how many assertions an identity provider can issue, or how many assertions can be stored.

13 Provide information for the Organization Profiles.

The optional Organization Profiles attributes contain values that define the organizational name of the entity.

Names
: Type the complete legal name of the entity's organization. Use the format *locale*|*organization-name*, for example, `en|`*organization-name*`.com`.

Note – If the Names attribute contains a value, it is required to add values to the Display Names and URL attributes.

Display Names
: Type a name that is suitable for display. Use the format *locale*|*organization-display-name*, for example, `en|`*organization-display-name*`.com`.

URL
: Type a URL that can be used to direct a principal to additional information on the entity's organization. Use the format *locale*|*organization-URL*, for example, `en|http://www.`*organization-name*`.com`.

14 Click New Contact Person to create a contact person for the provider.

The Contact Person attributes contain information regarding a human contact for the identity provider.

First Name
: Type the given name of the identity provider's contact person.

Last Name
: Type the surname of the identity provider's contact person.

Type
: Choose the contact's role from the drop-down menu:

- Administrative
- Billing
- Technical
- Other

Company
: Type the name of the company that employs the contact person.

Liberty Principal Identifier
: Type the name identifier that points to an online instance of the contact person's personal information profile.

Emails
: Type one or more email addresses for the contact person.

Telephone Numbers
: Type one or more telephone numbers for the contact person.

15 Click Create to create the contact person.

16 Click Save to complete the configuration, or define values for General or Identity Provider attributes by choosing from the View menu:

- **To define values for General attributes, see "To Configure General Attributes for a Provider Entity" on page 73.**
- **To define values for Identity Provider attributes, see "To Configure Hosted or Remote Identity Provider Attributes for a Provider Entity" on page 75.**

Configuring Affiliate Entities

After you create an affiliate entity, you populate it with affiliation information. This section contains the following procedures:

▼ To Configure an Affiliate Entity

1 In the Access Manager Console, select the Federation tab.

2 Under Federation, select the Entities tab.

3 Select the provider entity that you want to configure.

Ensure that you select an entity marked as type Affiliate.

4 Define values for the General or Affiliate attribute groupings by choosing from the View menu:

- **To define values for General attributes, see "To Configure General Attributes for an Affiliate Entity" on page 90**
- **To define values for Affiliate attributes, see "To Configure Affiliate Attributes for an Affiliate Entity" on page 92**

▼ To Configure General Attributes for an Affiliate Entity

Before performing this procedure, you must have completed the steps in "To Configure an Affiliate Entity" on page 90.

1 Choose General from the View menu, and provide information for the Entity Common Attributes.

Entity Common Attributes contain values that define the entity.

Entity Name
: The static value of this attribute is the name that you provided when creating the entity.

Type
: The static value of this attribute is Provider.

Description
: The value of this optional attribute is the description that you provided when creating the entity. You can modify the description.

Valid Until
: Type the expiration date for the entity metadata. Use Coordinated Universal Time (UTC) in the format *yyyy-mm-dd*T*hh:mm:ss.S*Z, for example, `2004-12-31T14:30:00.0Z`.

Cache Duration
Type the maximum amount of time that the entity metadata can be cached. Use the format P*n*Y*n*M*n*DT*n*H*n*M*n*S, where *n* is an integer variable. For example, `P1Y2M4DT9H8M20S` defines the cache duration as 1 year, 2 months, 4 days, 9 hours, 8 minutes, and 20 seconds.

2 Provide information for the Entity Contact Person Profile attributes.

Entity Contact Person Profile attributes contain values that define the administrator of the entity.

First Name
Type the given name of the entity's contact person.

Last Name
Type the surname of the entity's contact person.

Type
Choose the type of contact from the drop-down menu:

- Administrative
- Billing
- Technical
- Other

Company
Type the name of the company that employs this person.

Liberty Principal ID
Type a URI that points to an online instance of the contact person's personal information profile.

Emails
Type one or more email addresses for the contact person.

Telephone Numbers
Type one or more telephone numbers for the contact person.

3 Provide information for the Organization Profiles.

The optional Organization Profiles attributes contain values that define the organizational name of the entity.

Names
Type the complete legal name of the organization. Use the format *locale*|*organization-name*, for example, `en`|*organization-name*`.com`.

Note – If the Names attribute contains a value, it is required to add values to the Display Names and URL attributes also.

Display Names
Type a name that is suitable for display to a principal. The value is defined in the format *locale*|*organization-display-name*. For example, `en`|*organization-display-name*`.com`.

URL
Type a URL that can be used to direct a principal to additional information on the entity. Use the format *locale*|*organization-URL*, for example, `en|http://www.`*organization-name*`.com`.

4 Click Save to complete the configuration, or choose Affiliate from the View menu to configure the Affiliate attributes.

To define values for Affiliate attributes, see "To Configure Affiliate Attributes for an Affiliate Entity" on page 92.

▼ To Configure Affiliate Attributes for an Affiliate Entity

Before performing this procedure, you must have completed the steps in "To Configure an Affiliate Entity" on page 90.

1 Choose Affiliate from the View menu and provide information for the Common Attributes.

Common Attributes contain values that generally define the affiliation.

Name
The value of this attribute is the name of the affiliation.

Owner
The value of this attribute is the owner of the affiliation.

Valid Until
Type the expiration date for the affiliation metadata. Use Coordinated Universal Time (UTC) and the format *yyyy-mm-dd*T*hh:mm:ss.S*Z, for example, `2004-12-31T14:30:00.0Z`.

Cache Duration
Type the maximum amount of time affiliation metadata can be cached. Use the format P*n*Y*n*M*n*DT*n*H*n*M*n*S, where *n* is an integer. For example, `P1Y2M4DT9H8M20S` defines the cache duration as 1 year, 2 months, 4 days, 9 hours, 8 minutes, and 20 seconds.

Signing Key: Key Alias
Type the key alias that is used to sign requests and responses.

Encryption Key: Key Alias
Type the security certificate alias. Certificates are stored in a JKS keystore file. Each specific certificate is mapped to an alias that is used to fetch the certificate.

Encryption Key: Key Size
Type the length for keys used by the web service consumer when interacting with another entity.

Encryption Key: Encryption Method
Select the method of encryption:

- None
- AES
- DESede

2. **Select any of the available provider entities to assign as members of the affiliation.**

   A provider can belong to one or more affiliations. However, a provider without a specified authentication domain cannot participate in Liberty-based communications. Also, be sure that the service provider entity being assigned to the affiliate entity has enabled affiliation federation.

3. **Click Save to complete the configuration.**

4. **Click OK to complete the configuration, or choose General from the View menu to configure the General attributes.**

   To define values for General attributes, see "To Configure General Attributes for an Affiliate Entity" on page 90.

Deleting Entities

If an entity is to be deleted from the console, it first needs to be manually removed from the Trusted Providers list (if the provider is hosted) or the Available Providers list (if part of an affiliation).

▼ To Delete a Provider or Affiliate Entity

1. **In the Access Manager Console, click the Federation tab.**

2. **Under Federation, select the Entities tab.**

3. **Select the check box next to the entity that you want to delete.**

   No warning message is displayed when performing a delete.

4. **Click Delete.**

Authentication Domains

An *authentication domain* is a federation of any number of service providers (and at least one identity provider) with whom principals can transact business in a secure and apparently seamless environment. (The members of the domain must have previously established a *circle of trust* based on the Liberty Alliance Project architecture and operational agreements.)

Note – An authentication domain is *not* a domain in the domain name system (DNS) sense of the word.

The following procedures describe how to create, configure, and delete authentication domains using the Access Manager Console.

- "To Create An Authentication Domain" on page 94
- "To Configure or Modify an Authentication Domain" on page 94

- "To Delete an Authentication Domain" on page 95

To Create An Authentication Domain

1. **In the Access Manager Console, click the Federation tab.**
2. **Under Federation, select the Authentication Domains tab.**
3. **Select New.**

   The New Authentication Domain attributes are displayed.
4. **Type a name for the authentication domain.**
5. **(Optional) Type a description of the authentication domain in the Description field.**
6. **(Optional) Type a value for the Writer Service URL.**

   The Writer Service URL specifies the location of the service that writes the common domain cookie. Use the format `http://`*common-domain-host:port*`/common/writer`. For more information about the Common Domain Services, see Chapter 4.
7. **(Optional) Type a value for the Reader Service URL.**

   The Reader Service URL specifies the location of the service that reads the common domain cookie. Use the format `http://`*common-domain-host:port*`/common/transfer`. For more information about the Common Domain Services, see Chapter 4.
8. **Select Active or Inactive.**

   The default status is Active. Selecting Inactive disables communication within the authentication domain.
9. **Click OK.**

   The new authentication domain is now displayed in the list of configured Authentication Domains.

To Configure or Modify an Authentication Domain

1. **In the Access Manager Console, click the Federation tab.**
2. **Under Federation, select the Authentication Domains tab.**

   All created Authentication Domains are displayed.
3. **Click the name of the authentication domain that you want to modify.**

   The General and Providers properties for the authentication domain are displayed.
4. **(Optional) Enter or modify a description of the authentication domain in the Description field.**

5 **(Optional) Enter or modify the value for the Writer Service URL.**

The Writer Service URL specifies the location of the service that writes the common domain cookie. Use the format `http://`*common-domain-host:port*`/common/writer`. For more information on the Common Domain Services, see Chapter 4.

6 **(Optional) Enter or modify the value for the Reader Service URL.**

The Reader Service URL specifies the location of the service that reads the common domain cookie. Use the format `http://`*common-domain-host:port*`/common/transfer`. For more information on the Common Domain Services, see Chapter 4.

7 **Select Active or Inactive.**

The default status is Active. Selecting Inactive disables communication within the authentication domain.

8 **Click Add to populate the authentication domain with providers.**

The Trusted Providers page is displayed.

9 **Choose from the list of Available Providers and click Add.**

10 **Click OK to save the providers to the authentication domain.**

The authentication domain's attribute page is displayed.

11 **Click Save to complete the configuration.**

▼ To Delete an Authentication Domain

Deleting an authentication domain does not delete the providers that belong to it although it will impact the trusted relationship.

1 **In the Access Manager Console, click the Federation tab.**

2 **Under Federation, select the Authentication Domains tab.**

All created Authentication Domains are displayed.

3 **Select the check box next to the authentication domain that you want to delete.**

4 **Click Delete.**

Auto-Federation

The auto-federation feature in Access Manager will automatically federate a user's disparate provider accounts based on a common attribute. This common attribute will be exchanged in a single sign-on assertion so that the consuming service provider can identify the user and create account federations. If auto-federation is enabled and it is deemed that a user at provider A and a user at provider B have the same value for the defined common attribute (for example, `emailaddress`), the two accounts will be federated automatically without principal interaction.

Note – Auto-federating a principal's two distinct accounts at two different providers requires each provider to have agreed to implement support for this functionality beforehand.

▼ To Enable Auto Federation

Ensure that each local service and identity provider participating in auto federation is configured for it. Remote providers would not be configured in your deployment.

1. **In the Access Manager Console, click the Federation tab.**
2. **Under Federation, select the Entities tab.**
3. **Select the name of a hosted provider entity to edit its profile.**

   Whether an entity is configured to hold hosted or remote providers is not information that is disclosed on this screen.
4. **Select Identity Provider or Service Provider from the View menu.**
5. **Select Access Manager Configuration.**
6. **Enable Auto Federation by checking the box.**
7. **Type a value for the Auto Federation Common Attribute Name attribute.**

   For example, enter `emailaddress` or `userID`. You should be sure that each participating user profile (at both providers) has a value for this attribute.
8. **Click Save to complete the configuration.**

Bulk Federation

Access Manager provides a script for federating user accounts in bulk. It is called `ambulkfed` and is located in `/opt/SUNWam/bin`. The script assumes that the user database is LDAPv3–compliant.

Note – The `ambulkfed` script is the primary script for bulk federation. It uses two other Perl scripts, `amGenerateLDIF.pl` and `amGenerateNI.pl`, behind the scenes.

As input, the script takes a file that maps the user distinguished name (DN) of the identity provider to the user DN of the service provider. Each line of the file must place the mappings in the following order and separated by a pipe ("|"): `uid=spuser,dc=iplanet,dc=com | uid=idpuser,dc=iplanet,dc=com`. The script generates unique random identifiers for each mapping and creates four files:

- `spnameidentifiers.txt`
- `idpnameidentifiers.txt`
- `spuserdata.ldif`
- `idpuserdata.ldif`

These files contain the data for bulk federation. The LDIFs are used for instances of Access Manager. `ambulkfed` generates and loads the LDIF data into Access Manager based on its given provider role. For example, it will load `spuserdata.ldif` if Access Manager acts as a service provider and it will load `idpuserdata.ldif` if Access Manager acts as an identity provider. The LDIFs will also be stored locally and can be used with `ldapmodify` to load the data into a remote instance of Access Manager. If the remote provider is not an instance of Access Manager, the generated text files `spnameidentifiers.txt` and `idpnameidentifiers.txt` can be used to generate federation data based on the input needs of the provider.

Dynamic Identity Provider Proxying

An identity provider that is asked to authenticate a principal that has already been authenticated with another identity provider may proxy the authentication request, on behalf of the requesting service provider, to the authenticating identity provider. This is called *dynamic identity provider proxying*. When the first identity provider receives an authentication request regarding a principal, it prepares a new authentication assertion on its own behalf by referencing the relevant information from the original assertion and sending the assertion to the authenticating identity provider.

Note – The service provider requesting authentication may control this proxy behavior by setting a list of preferred identity providers or by defining the amount of times the identity provider can proxy the request.

▼ To Configure and Test Dynamic Identity Provider Proxying

The following steps describe the procedure to enable three machines for identity provider proxying and test the configuration. The procedure assumes the three machines have Access Manager installed and are configured as follows:

Machine	Authentication Function	Federation Function
Machine 1	Authenticating Identity Provider	Identity Provider
Machine 2	Proxying Identity Provider	Identity Provider and Service Provider
Machine 3	Requesting Service Provider	Service Provider

All of the WAR files and metadata used in the following procedure can be found in */AccessManager-base*/`samples/liberty/sample1`.

1 To configure machine 3, deploy the SP1 WAR files and load `sp1Metadata.xml`.

Ensure that the metadata defines machine 2 as an identity provider and machine 3 as a service provider.

2 To configure machine 1, deploy the IDP1 WAR files and load `idp1Metadata.xml`.

Ensure that the metadata defines machine 1 as an identity provider and machine 2 as a service provider.

3 To configure machine 2, do the following:

a. To configure machine 2 as a service provider, deploy the SP1 WAR files.

Modify `AMClient.properties` to reflect this.

b. To configure machine 2 as an identity provider, load a second, modified `idp1Metadata.xml`.

Ensure that `idp1Metadata.xml` contains *only* data that defines machine 1 as an identity provider. Remove all other metadata.

4 Log in to machine 2 and modify the following metadata:

a. Change the value of the Authentication Type attribute to Local.

This attribute can be found in the Access Manager Configuration section of the entity describing machine 2 as a service provider.

b. Add machine 1 and machine 3 to the list of Trusted Providers configured for machine 2.

This attribute can be found in the Trusted Provider section of the entity describing machine 2 as a service provider.

c. **Save the configuration.**

5 **Also on machine 2, modify the following metadata regarding machine 3.**

a. **Select the check box next to Enable Proxy Authentication.**

This attribute can be found in the Proxy Authentication Configuration section of the entity that defines machine 3 as an identity provider.

b. **Add machine 1 to the list of Proxy Identity Providers List.**

This attribute can be found in the Proxy Authentication Configuration section of the entity that defines machine 3 as an identity provider. The value is a URI defined as the provider's identifier.

c. **Set Maximum Number of Proxies to 1.**

d. **Save the configuration.**

6 **Federate a user between machine 3 (acting as a service provider) and machine 2 (acting as an identity provider).**

7 **Federate a user between machine 2 (acting as a service provider) and machine 1 (acting as an identity provider).**

8 **Close the browser and attempt single sign-on.**

You will be redirected to machine 1 rather than machine 2 if you enter the username and password used to federate with machine 1.

The Pre-login URL

The pre-login process is the entry point for applications participating in Liberty-based single sign-on. As described in "Process of Federation" on page 64, the principal would be redirected to the location defined by the pre-login URL if no Access Manager session token is found. This default process, though, can be modified based on the values of URL query parameters passed to Access Manager by the service provider.

Note – A URL parameter is a name/value pair appended to the end of a URL. The parameter starts with a question mark (?) and takes the form *name*=*value*. A number of parameters can be combined in one URL although if more than one parameter exists, they are separated by an ampersand (&).

In order to modify the pre-login URL, edit the property in either the `AMConfig.properties` file or the `AMAgent.properties` file, dependant on your deployment. Use the format `http://`*hostname*`:`*port*`/`*deploy-uri*`/preLogin?metaAlias=`*metaAlias*. Query parameters can be appended to the URL as &*param1*=*value1*&*param2*=*value2* and so on. These parameters and their usage and values are described in the following table.

TABLE 3–2 Pre-login URL Parameters for Federation

Parameter	Description
`actionOnNoFedCookie`	The `actionOnNoFedCookie` parameter provides the flexibility to redirect a user when the `fedCookie` is not present in the browser, and when there is only one identity provider. It takes the following values: ■ `commonlogin` will redirect to a common login page. ■ `locallogin` will redirect to the local Access Manager login page. ■ `passive` will issue a request to the identity provider by setting the `isPassive` parameter of the `AuthnRequest` element to `true`. ■ `active` will issue a normal single sign-on request to the identity provider.
`anonymousOnetime`	The `anonymousOnetime` parameter can be used by service providers that authenticate users with anonymous, one time federation sessions. A value of `true` enables the service provider to issue a one time federation request and generate an anonymous session after successful verification of the authentication assertion from the identity provider. This feature is useful when the service provider doesn't have a user repository (for example, `http://www.weather.com`) but would like to depend on an identity provider for authentication. When the service provider receives a successful authentication assertion from an identity provider, they would generate an anonymous, temporary session.
`authlevel`	The `authlevel` parameter takes as a value a positive number that maps to an authentication level defined in the Access Manager Authentication Framework. The authentication level indicates how much to trust a method of authentication. **Note –** More information on the authentication framework can be found in *Sun Java System Access Manager 7 2005Q4 Administration Guide*. In this framework, each service provider is configured with a default authentication context (preferred method of authentication). However, the provider might like to change the assigned authentication context to one that is based on the defined authentication level. For example, provider B would like to generate a local session with an authentication level of 3 so it requests the identity provider to authenticate the user with an authentication context assigned that level. The value of this query parameter determines the authentication context to be used by the identity provider.
`gotoOnFedCookieNo`	The `gotoOnFedCookieNo` parameter takes as a value a URL to which the principal is redirected if a `fedCookie` with a value of `no` is found. The default behavior is to redirect the user to the Access Manager login page.

Federation API

The `com.sun.liberty` package provides the interface that forms the basis of the Federation API. It contains the `LibertyManager` class which must be instantiated by web applications that want to access the Federation component. 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. For more detailed information, see the Java API Reference in */AccessManager-base*`/SUNWam/docs` or on docs.sun.com.

TABLE 3–3 Federation API Methods

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

Sample Federation Environment

Access Manager provides a collection of samples based on the Liberty Alliance Project specifications. They are located in the */AccessManager-base*`/SUNWam/samples/liberty/` directory. Appendix A includes information about these samples.

Sample 1, located in */AccessManager-base*`/SUNWam/samples/liberty/Sample1`, can be used to configure an environment for creating and managing a federation. The sample demonstrates the basic use of various Liberty-based federation protocols including account federation, single sign-on, single logout, and federation termination. Completing the procedures in the sample `Readme.txt` or `Readme.html` will help to give you a more complete understanding of how federation works.

Note – The `Readme` file also contains instructions for configuring a common domain. For information about common domains, see Chapter 4.

◆ ◆ ◆ CHAPTER 4

Common Domain Services

Sun Java System Access Manager provides Common Domain Services that enable a service provider to determine the identity provider used by a principal in an authentication domain that contains multiple identity providers.

This chapter covers the following topics:

Common Domain

Providers need a way to find which identity provider is used by a principal requesting authentication. Because authentication domains are configured without regard to their location, this function must work across DNS—defined domains. Suppose an authentication domain contains more than one identity provider, then a service provider in the authentication domain trusts more than one identity provider. But, a principal can only issue a federation request to one identity provider, so the service provider to which the principal is requesting access must have the means to determine the correct one. To ascertain a principal's identity provider, the service provider invokes a protocol exchange to retrieve the *common domain cookie*, a cookie written for the purpose of *introducing* the identity provider to the service provider. If no common domain cookie is found when the principal issues a federation request, the service provider must present a list of trusted identity providers from which the principal will choose. After successful authentication, the identity provider writes (using the Writer Service URL as defined in "Configuring the Common Domain Services URLs" on page 105) a common domain cookie. The next time the principal attempts to access a service, the service provider finds and reads the common domain cookie (using the Reader Service URL as defined in "Configuring the Common Domain Services URLs" on page 105), to determine the identity provider.

The *common domain* is established for use only within the scope of the Common Domain Services. In Access Manager deployments, the Common Domain Services are deployed in a web container installed in a predetermined and pre-configured *common* domain so that the common domain cookie is accessible to all providers in the authentication domain. If the HTTP server in the common domain is operated by the service provider, the service provider will redirect the user agent to the identity provider.

After a principal authenticates with a particular identity provider, the identity provider redirects the principal's browser to their Common Domain Service with a parameter indicating that they are the identity provider for this principal. The Common Domain Service then writes a cookie using that preference. Thereafter, all providers configured in this common domain will be able to tell which identity provider is used by the principal. For example, suppose an identity provider is available at `http://www.Bank.com` and a service provider is available via `http://www.Store.com`. If the common domain they define is `RetailGroup.com`, the addresses will be `Bank.RetailGroup.com` and `Store.RetailGroup.com`, respectively.

Note – The Common Domain Services are based on the Identity Provider Introduction Profile detailed in the *Liberty ID-FF Bindings and Profiles Specifications*.

Common Domain Cookie

After an identity provider authenticates a principal, the identity provider sets a URL-encoded cookie that is defined in a predetermined domain common to all identity providers and service providers within the authentication domain. The *common domain cookie* is named `_liberty_idp`. After successful authentication, a principal's identity provider appends their encoded identifier to a list in the cookie. If their identifier is already present in the list, the identity provider may remove the initial appearance and append it again. The intent is that the service provider reads the last identifier on the cookie's list to find the principal's most recently established identity provider.

The identifiers in the common domain cookie are a list of `SuccinctID` elements encoded in the Base64 format. One element maps to each identity provider in the authentication domain. Service providers then use this `SuccinctID` element to find the user's preferred identity provider.

Note – When the request contains no common domain cookie, the service provider presents a list of trusted identity providers from which the principal may choose.

Configuring the Common Domain Services URLs

In Access Manager, the Common Domain Services are configured using two URLs that point to servlets developed for writing and reading the common domain cookie. They are:

- "Writer Service URL" on page 105
- "Reader Service URL" on page 105

Note – For more information on how to configure these URLs, see "To Create An Authentication Domain" on page 94 in Chapter 3.

Writer Service URL

The Writer Service URL is used by the identity provider. After successful authentication, the common domain cookie is appended with the query parameter `_liberty_idp=`*entity-ID-of-identity-provider*. This parameter is used to redirect the principal to the Writer Service URL defined for the identity provider. The URL is configured as the value for the Writer Service URL attribute when an authentication domain is created. Use the format `http://`*common-domain-host:port*`/`*deployment-uri*`/writer` where *common-domain-host:port* refers to the machine on which the Common Domain Services are installed and *deployment-uri* tells the web container where to look for information specific to the application (such as classes or JARs). The default URI is `amcommon`.

Reader Service URL

The Reader Service URL is used by the service provider. The service provider redirects the principal to this URL in order to find the preferred identity provider. Once found, the principal is redirected to the identity provider for single sign-on. The URL is defined as the value for the Reader Service URL attribute when an authentication domain is created. It is formatted as `http://`*common-domain-host:port*`/`*deployment-uri*`/transfer` where *common-domain-host:port* refers to the machine on which the Common Domain Services are installed and *deployment-uri* tells the web container where to look for information specific to the application (such as classes or JARs). The default URI is `amcommon`.

Configuring the Common Domain Services Properties

`FSIntroConfig.properties` is a file that contains properties used to configure the Common Domain Services. It is deployed as part of the web application and located in */*AccessManager-base*`/web-src/common/WEB-INF/classes`. It contains the properties described in the following table (which may be modified).

TABLE 4–1 Common Domain Services Properties in `FSConfig.properties`

Property	Description
`com.sun.identity.federation.services.introduction.cookiedomain`	The value of this property is the name of the common domain.
`com.sun.identity.federation.services.introduction.cookietype`	This property takes a value of either `PERSISTENT` or `SESSION`. `PERSISTENT` defines the cookie as one that will be stored and reused after a web browser is closed and reopened. `SESSION` defines the cookie as one that will not be stored after the web browser has been closed.
`com.iplanet.am.cookie.secure`	This property takes a value of either `false` or `true`. It defines whether the cookie needs to be secured or not.
`com.iplanet.am.cookie.encode`	This property takes a value of either `false` or `true`. It defines whether the cookie will be URL encoded or not. This property is useful if, for example, the web container that reads or writes the cookie decrypts or encrypts it by default.

Installing the Common Domain Services for Federation

The Common Domain Services are installed as a web application within Access Manager using the Sun Java Enterprise System installer. However, the Common Domain Services for Federation can also be installed as a standalone web application (separate from the Access Manager product) on a Java Enterprise Edition web container. This option allows for generating common domain cookies on a machine on which Access Manager is not installed. Once the Common Domain Services for Federation is installed, you must set up the writer URL attribute for any identity providers and the reader URL for any service providers. These URLs point to the machine on which Common Domain Services for Federation is installed. For more information, see the *Sun Java Enterprise System 2005Q4 Installation Guide for UNIX*.

Tip – In most real world deployments, installing the Common Domain Services on a separate machine is the obvious choice because of the need to setup a third-level common domain between service providers and identity providers in disparate enterprises.

▼ To Test a Common Domain Services Installation

For troubleshooting, make sure the debug level property in `FSIntroConfig.properties` is set to `message`.

1. **Install the Common Domain Services for Federation as a standalone application in a web container in the common domain.**

   Ensure that the common domain has been defined and the web container is installed in it.

2. **Modify the properties in `FSIntroConfig.properties` as needed.**

   See "Configuring the Common Domain Services Properties" on page 105 for more information.

3. **Configure at least two identity providers for a service provider.**

   Ensure that the "Writer Service URL" on page 105 is configured for each identity provider and the "Reader Service URL" on page 105 is configured for each service provider.

4. **Login as a user and complete federation and single sign-on between one identity provider and the service provider.**

   En sure that the `_liberty_idp` cookie is set to the common domain.

5. **Login as a user and complete federation and single sign-on between the second identity provider and the service provider.**

   After the initial successful federation and single sign-on, all service providers in the common domain are redirected to the first identity provider based on the information in the common domain cookie.

Note – Whether the cookie is persistent or for this browser session alone is dependent on how `FSIntroConfig.properties` is configured.

PART III

Supported Web Services

◆ ◆ ◆ CHAPTER 5

Authentication Web Service

Sun Java™ System Access Manager contains an implementation of the *Liberty ID-WSF Authentication Service Specification* developed by the Liberty Alliance Project. The implementation of the specifications is called the Authentication Web Service which allows authentication using SOAP messages.

This chapter covers the following topics:

Authentication Web Service Overview

The implementation of the Access Manager Authentication Web Service is based on the *Liberty ID-WSF Authentication Service Specification*. The specification defines a protocol that adds authentication functionality to the SOAP binding discussed in the *Liberty ID-WSF SOAP Binding Specification* and, Chapter 8 in this guide. The specification also contains an XML schema that defines the authentication protocol.

Note – This XML Schema Definition (XSD) file can be found on the Liberty Alliance Project web site. Version 1.0 is also reproduced in Appendix B.

The Authentication Web Service is for provider-to-provider authentication. The Simple Authentication and Security Layer (SASL) is the method used to add this authentication support.

XML Service File

The Authentication Web Service is configured using the XML service file `amAuthnSvc.xml`. This file defines the attribute for the Authentication Web Service which can be managed through the Access Manager console or the XML file.

Note – For information about service files, see the *Sun Java System Access Manager 7 2005Q4 Administration Guide*.

Authentication Web Service APIs

The Access Manager Authentication Web Service includes the following Java programming packages:

- `com.sun.identity.liberty.ws.authnsvc`
- `com.sun.identity.liberty.ws.authnsvc.mechanism`
- `com.sun.identity.liberty.ws.authnsvc.protocol`

The first package is a client API for external Java applications to send SASL requests and receive SASL responses. The second package defines an interface to handle different SASL mechanisms. The final package contains classes that represent the SASL request and response. Together, the packages are used to initiate the authentication process and communicate authentication credentials to the Authentication Web Service. For more information, see the "Authentication Web Service API" on page 116.

Which Authentication Service to Use?

The Liberty-based Authentication Web Service is not to be confused with the proprietary Authentication Service discussed in the *Sun Java System Access Manager 7 2005Q4 Administration Guide*. Architecturally, the Authentication Web Service sits on top of the Access Manager Authentication Service and the Liberty Alliance Project framework. You might use the Authentication Web Service if you are a service provider and want to use a standards-based mechanism to authenticate users.

Following are two use cases where the Authentication Web Service is preferable to the Access Manager Authentication Service:

- A service provider relies on a remote identity provider (not necessarily using Access Manager) for authentication.
- An enterprise in a service-oriented architecture (SOA) environment wants to use nonproprietary mechanisms to authenticate users and web services clients before accessing a protected web service.

In addition to providing an authentication service to verify credentials (for example, user ID and password), the Authentication Web Service provides the web services consumer (WSC) with

bootstrap information that contains the endpoint and credentials needed to access the Discovery Service (as discussed in Chapter 7). The WSC can ignore the bootstrap or use it to access other web services, such as the authenticated user's personal profile or calendar.

Note – An authenticated enterprise might also use the bootstrap information to access a partner in a business-to-business environment.

Following is an example that shows how the Authentication Web Service interacts with the Access Manager Authentication Service. It assumes the following separate entities:

- A user (principal)
- A service provider (acting as a WSC)
- An identity provider hosted by Access Manager where the Access Manager Authentication Service is configured for Certificate and LDAP authentication and the Authentication Web Service has mapped LDAP to its own PLAIN authentication mechanism
- The user's personal profile (hosted by another product)

The WSC delegates all authentication to the identity provider and prefers PLAIN authentication which accepts a user identifier and password.

1. The user attempts access to a service provider (not necessarily hosted by Access Manager).
2. When the service provider finds that the user is not authenticated, it invokes the identity provider's Authentication Web Service by sending it a SOAP request.
3. After inspecting its configuration, the Authentication Web Service sends back a response indicating that it supports Certificate and PLAIN authentication.
4. The service provider decides on PLAIN authentication and prompts the user for an identifier and password.
5. Interactions based on the standard PLAIN authentication mapping ensues between the service provider and identity provider (hosted on Access Manager) using the Authentication Web Service.
    a. The service provider receives the user identifier and password and sends it to the identity provider.
    b. The identity provider passes the credentials to the locally hosted LDAP Authentication module using the proprietary Access Manager Authentication Service's Java API.
    c. The LDAP Authentication module verifies the credentials.
    d. The Authentication Web Service is notified of the verification and sends a response to the service provider indicating successful authentication. If configured to do so, it also includes bootstrap information formatted using XML and containing the Discovery Service endpoint and a token to invoke it.
6. The service provider parses the response, verifies that it is a successful authentication, and provides the service to the user.

At some point the service provider might need to access the user's personal profile. To do this, it will use the bootstrap information received during this process to contact the Discovery Service and find where the profile is stored. The Discovery Service returns a *resource offering* (containing the location of an endpoint and a token), and the service provider uses that to invoke the Liberty Personal Profile Service.

Authentication Web Service Process

The exchange of authentication information between a web service consumer (WSC) and the web service provider (WSP) is accomplished using SOAP-bound messages. The messages are a series of client requests and server responses specific to the defined SASL mechanism (or mode of authentication).

The authentication exchange can involve an arbitrary number of round trips, dictated by the particular SASL mechanism employed. The WSC might have knowledge of the supported SASL mechanisms, or it might send the server its own list of mechanisms and allow the server to choose one. The list of supported mechanisms can be found at `http://www.iana.org/assignments/sasl-mechanisms`.

After receiving a request for authentication (or any response from the WSC), the WSP may issue additional challenges or indicate authentication failure or success. The sequence between the WSC and the Authentication Web Service (a WSP) is as follows.

1. The authentication exchange begins when a WSC sends an SASL authentication request to the Authentication Web Service on behalf of a principal.

   The request message contains an identifier for the principal and indicates one or more SASL mechanisms from which the service can choose.
2. The Authentication Web Service responds by asserting the method to use and, if applicable, initiating a challenge.

   If the Authentication Web Service does not support any of the cited methods, it responds by aborting the exchange.
3. The WSC responds with the necessary credentials for the chosen method of authentication.
4. The Authentication Web Service replies by approving or denying the authentication.

   If approved, the response includes the credentials the WSC needs to invoke other web services, such as the Discovery Service.

Authentication Web Service Attribute

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.

Note – For information about the types of attributes used in Access Manager, see the *Sun Java System Access Manager 7 2005Q4 Technical Overview*.

The attribute for the Authentication Web Service is defined in the `amAuthnSvc.xml` service file and is called the Mechanism Handlers List.

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 5–1 Default Implementations for Authentication Mechanism

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

Note – The Authentication Web Service layer provides an interface that must be implemented for each SASL mechanism to process the requested message and return a response. For more information, see "`com.sun.identity.liberty.ws.authnsvc.mechanism` Package" on page 116.

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 116
- "com.sun.identity.liberty.ws.authnsvc.mechanism Package" on page 116
- "com.sun.identity.liberty.ws.authnsvc.protocol Package" on page 116

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*. This schema is reproduced in Appendix B.

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.

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

◆ ◆ ◆ CHAPTER 6

Data Services

Sun Java System Access Manager contains implementations of the *Liberty ID-WSF Data Services Template Specification* in addition to instructions on how you can add a custom data service to your deployment.

This chapter covers the following topics:

Data Services Overview

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.

Note – To develop your own data service see the instructions in "Developing A New Data Service" on page 131.

Liberty ID-WSF Data Services Template Specification

The Liberty ID-WSF-DST specifies a base layer that can be extended by any instance of a data service. An example of a data service is an identity service, such as an online corporate directory. When you want to contact a colleague, you conduct a search based on the individual's name, and the data service returns information associated with that person's identity. The information might include the individual's office location and phone number, as well as job title or department name. For proper implementation, all data services must be built on top of the Liberty ID-WSF-DST because it provides the data model and message interfaces. The following figure illustrates how Access Manager uses the Liberty ID-WSF-DST as the framework for data services.

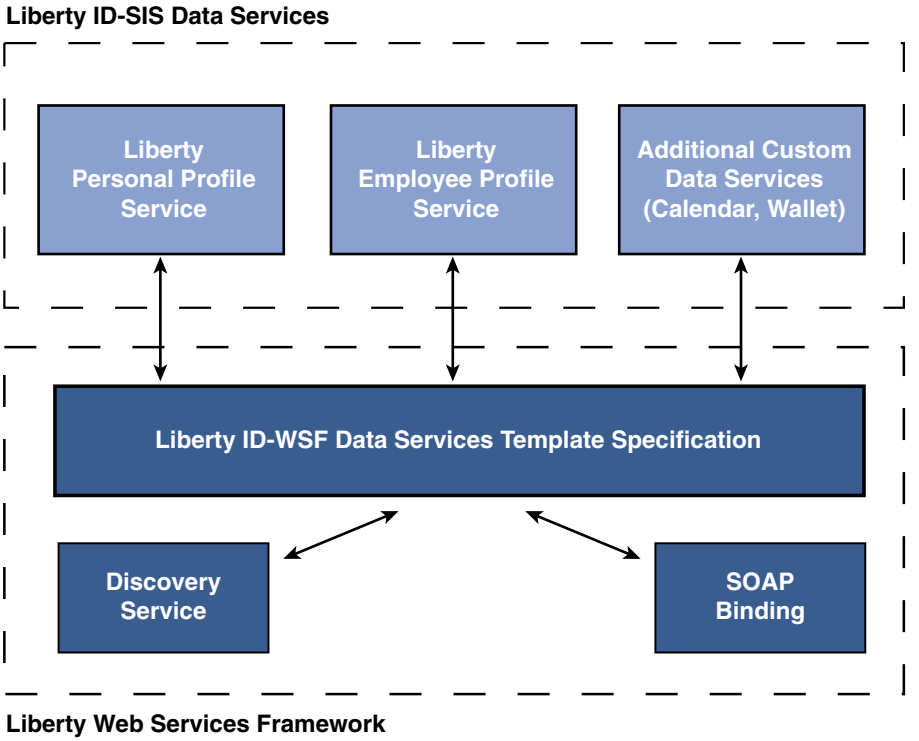

FIGURE 6–1 Data Service Template as Building Block of Data Services

The Web Services framework in Access Manager uses the Liberty ID-WSF-DST to develop data services. The Access Manager Liberty Personal Profile Service and Liberty Employee Profile Service were developed on top of the Web Services framework, using the specification. Additional data services can also be developed by the customer.

Note – For more information on the data services specification, see the *Liberty ID-WSF Data Services Template Specification*.

Liberty Personal Profile Service

The *Liberty ID-SIS Personal Profile Service Specification* (Liberty ID-SIS-PP) describes a data service that provides an identity's basic profile information, such as full name, contact details, and financial information. This data service is intended to be the least common denominator for holding consumer-based information about a principal. Access Manager has implemented this specification and developed the Liberty Personal Profile Service.

For more information, see the *Liberty ID-SIS Personal Profile Service Specification*.

XML Service File

The Access Manager Liberty Personal Profile Service is configured using the XML service file `amLibertyPersonalProfile.xml`. This file defines attributes for the Liberty Personal Profile Service which can be managed through the Access Manager Console or the XML file itself.

Note – For information about service files, see the *Sun Java System Access Manager 7 2005Q4 Administration Guide*.

XSD Schema Definition

The Liberty ID-SIS-PP also defines an XML schema for use in building a personal profile service. This XML Schema Definition (XSD) file is on the Liberty Alliance Project web site. Version 1.0 is also reproduced in Appendix B.

Liberty Employee Profile Service

The *Liberty ID-SIS Employee Profile Service Specification* (Liberty ID-SIS-EP) describes a data service that provides an identity's profile information as it relates to employment. An example of a employee profile service might be a corporate calendar or phone book.

Access Manager has implemented this specification by developing a sample that includes the files needed to deploy and invoke a Liberty Employee Profile Service. The Liberty Employee Profile Service is not available when Access Manager is installed. It must first be deployed. For information about accessing the sample files and how to deploy them, see "Liberty Employee Profile Service" on page 129.

Note – For more information, see the *Liberty ID-SIS Employee Profile Service Specification*.

XML Service File

Among the files included with the sample is the XML service file `amLibertyEmployeeProfile.xml`. This file defines the attributes for the Liberty Employee Profile Service which, once deployed, can be managed through the Access Manager Console or the XML file itself.

Note – For information about service files, see the *Sun Java System Access Manager 7 2005Q4 Administration Guide*.

XSD Schema Definition

The Liberty ID-SIS-EP also defines an XML schema for use in building an employee profile service. This XSD file is on the Liberty Alliance Project web site. Version 1.0 is also reproduced in Appendix B.

Data Services API

Access Manager data services are built using a Java package called `com.sun.identity.liberty.ws.dst`. Access Manager provides this package for developing custom services based on the Liberty ID-WSF-DST. Additional information about these interfaces can be found in "Data Services Template API" on page 129 and in the Java API Reference at */AccessManager-base*`/SUNWam/docs` or on docs.sun.com.

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 Chapter 7.

Liberty Personal Profile Service Process

The invocation of a personal profile begins when a WSC posts a query or a modify request to the Liberty Personal Profile Service on behalf of a user. The following process is also illustrated in Figure 6–2.

1. A web services client uses the Data Services Template API to post a query or a modify request to the Liberty Personal Profile Service.

   All the query or modify requests to any identity service are SOAP requests.

2. The client's SOAP request is received by the SOAP receiver provided by the SOAP Binding Service.

The SOAP receiver invokes either the Discovery Service, the Authentication Web Service, or the Liberty Personal Profile Service, depending on the service key transmitted as part of the URL. The SOAP Binding Service might also authenticate the client identity.

3. The Liberty Personal Profile Service implements the `DSTRequestHandler` to process the request.

   The request is processed based on the request type (query or modify) and the query expression. Processing might entail the authorization of a WSC using the Access Manager Policy Service, or it might entail using the Interaction Service for interacting with the user before sending data to the WSC.

4. The Liberty Personal Profile Service builds a service response, adds credentials (if they are required), and sends the response back to the WSC.

   - For a response to a query request, the Liberty Personal Profile Service builds a personal profile container (as defined by the specification). It is formatted in XML and based on the Query Select expression. The Personal Profile attribute values are extracted from the data store by making use of the attribute mapper. The attribute mapper is defined by the XML service file, and the attribute values will be used while building the XML container. The Personal Profile Service then applies `xpath` queries on the XML and provides us with the resultant XML data node.
   - For a response to a modify request, the Liberty Personal Profile Service parses the Modifiable Select expression and updates the new data from the new data node in the request.

The following diagram illustrates the Liberty Personal Profile Service process.

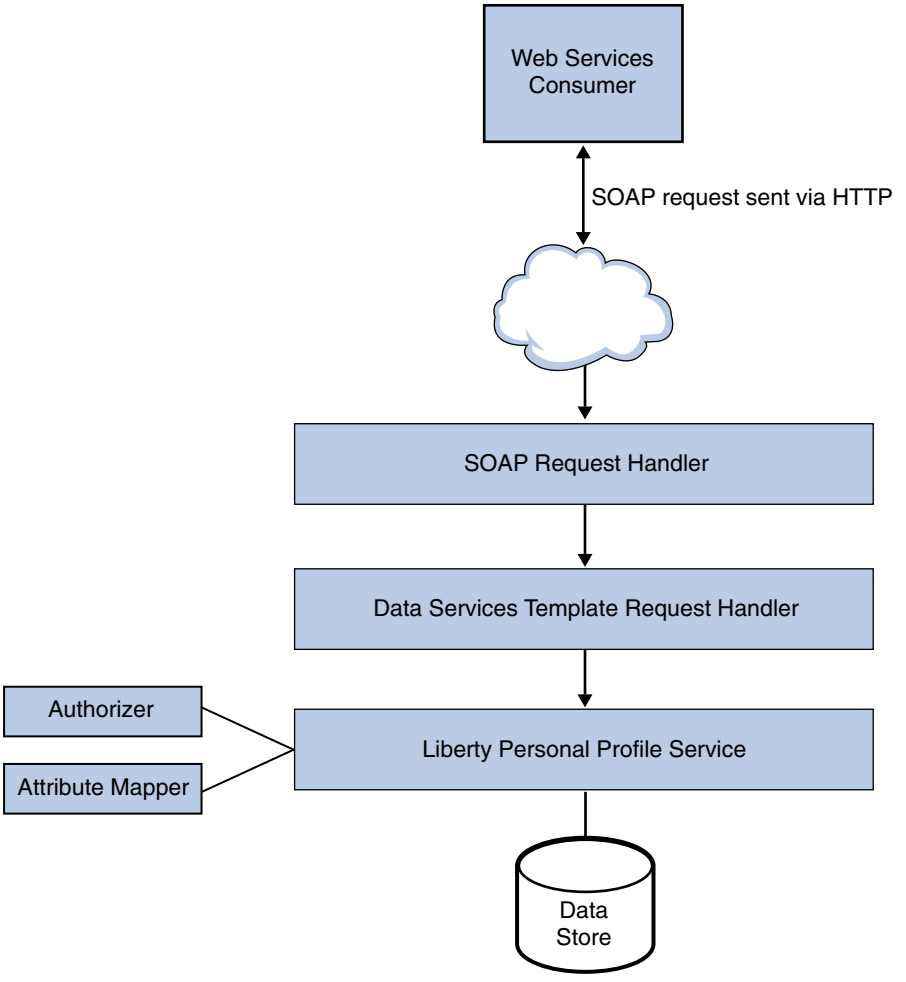

FIGURE 6–2 Liberty Personal Profile Service Process

Liberty Personal Profile Service Attributes

The Liberty Personal Profile Service attributes are *global* attributes. The values of these attributes are carried across the Access Manager configuration and inherited by each configured organization.

Note – For information about the types of attributes used in Access Manager, see the *Sun Java System Access Manager 7 2005Q4 Technical Overview*.

Attributes for the Personal Profile Service are defined in the `amLibertyPersonalProfile.xml` service file. The attributes are:

ResourceID Mapper

The value of this attribute specifies the implementation of `com.sun.identity.liberty.ws.interfaces.ResourceIDMapper`. Although a new implementation can be developed, Access Manager provides the default `com.sun.identity.liberty.ws.idpp.plugin.IDPPResourceIDMapper`, which maps a discovery resource identifier to a user identifier.

Authorizer

Before processing a request, the Liberty Personal Profile Service verifies the authorization of the WSC making the request. There are two levels of authorization verification:

- Is the requesting entity authorized to access the requested resource profile information?
- Is the requested resource published to the requestor?

Authorization occurs through a plug-in to the Liberty Personal Profile Service, an implementation of the `com.sun.identity.liberty.ws.interfaces.Authorizer` interface. Although a new implementation can be developed, Access Manager provides the default class, `com.sun.identity.liberty.ws.idpp.plugin.IDPPAuthorizer`. This plug-in defines four policy action values for the `query` and `modify` operations:

- `Allow`
- `Deny`
- `Interact For Consent`
- `Interact For Value`

The resource values for the rules are similar to `x-path` expressions defined by the Liberty Personal Profile Service. For example, a rule can be defined like this:

```
/PP/CommonName/AnalyzedName/FN    Query   Interact for consent
/PP/CommonName/*                  Modify  Interact for value
/PP/InformalName                  Query   Deny
```

Authorization can be turned off by deselecting one or both of the following attributes, which are also defined in the Liberty Personal Profile Service:

- "Require Query PolicyEval" on page 127
- "Require Modify PolicyEval" on page 127

Attribute Mapper

The value of this attribute defines the class for mapping a Liberty Personal Profile Service attribute to an Access Manager user attribute. By default, the class is `com.sun.identity.liberty.ws.idpp.plugin.IDPPAttributeMapper`.

Note – `com.sun.identity.liberty.ws.idpp.plugin.IDPPAttributeMapper` is not a public class.

Provider ID

The value of this attribute defines the unique identifier for this instance of the Liberty Personal Profile Service. Use the format *protocol*`://`*hostname*`:`*port*/*deloy-uri*`/Liberty/idpp`.

Name Scheme

The value of this attribute defines the naming scheme for the Liberty Personal Profile Service common name. Choose First Last or First Middle Last.

Namespace Prefix

The value of this attribute specifies the namespace prefix that is used for Liberty Personal Profile Service XML protocol messages. A *namespace* differentiates elements with the same name that come from different XML schemas. The Namespace Prefix is prepended to the element.

Supported Containers

The values of this attribute define a list of supported containers in the Liberty Personal Profile Service. A *container*, as used in this instance, is an attribute of the Liberty Personal Profile Service.

Note – The term *container* as described in this section is not related to the Access Manager identity-related object that is also called *container*.

For example, Emergency Contact and Common Name are two default containers for the Liberty Personal Profile Service. To add a new container, click Add, enter values in the provided fields and click OK.

Note – This functionality is not yet public.

PPLDAP Attribute Map List

Each identity attribute defined in the Liberty Personal Profile Service maps one-to-one with an Access Manager LDAP attribute. For example, `JobTitle=sunIdentityServerPPEmploymentIdentityJobTitle` maps the Liberty `JobTitle` attribute to the Access Manager `sunIdentityServerPPEmploymentIdentityJobTitle` attribute.

The value of this attribute is a list that specifies the mappings. The list is used by the attribute mapper defined in "Attribute Mapper" on page 126, by default, `com.sun.identity.liberty.ws.idpp.plugin.IDPPAttributeMapper`.

Note – When adding new attributes to the Liberty Personal Profile Service or the LDAP data store, ensure that the new attribute mappings are configured as values of this attribute.

In the following code sample, the Liberty Personal Profile Service `informalName` attribute mapping to the LDAP attribute `uid` is added to the mappings already present in the Liberty Personal Profile Service XML service file, `amLibertyPersonalProfile.xml`.

Note – Attribute mappings are defined as global attributes under the name `sunIdentityServerPPDSAttributeMapList` in `amLibertyPersonalProfile.xml`. This attribute corresponds to that `sunIdentityServerPPDSAttributeMapList` global attribute.

```
<AttributeSchema name="sunIdentityServerPPDSAttributeMapList"
          type="list"
          syntax="string"
          i18nKey="p108">
          <DefaultValues>
              <Value>CN=sunIdentityServerPPCommonNameCN</Value>
              <Value>FN=sunIdentityServerPPCommonNameFN</Value>
              <Value>MN=sunIdentityServerPPCommonNameMN</Value>
              <Value>SN=sunIdentityServerPPCommonNameSN</Value>
              <Value>InformalName=uid</Value>
          </DefaultValues>
</AttributeSchema>
```

Require Query PolicyEval

If selected, this option requires that a policy evaluation be performed for Liberty Personal Profile Service queries. For more information, see "Authorizer" on page 125.

Require Modify PolicyEval

If selected, this option requires that a policy evaluation be performed for Liberty Personal Profile Service modifications. For more information, see "Authorizer" on page 125.

Extension Container Attributes

The Liberty Personal Profile Service allows you to specify extension attributes that are not defined in the Liberty Alliance Project specifications. The values of this attribute specify a list of extension container attributes. All extensions should be defined as:

```
/PP/Extension/PPISExtension [@name='extensionattribute']
```

The following sample illustrates an extension query expression for `creditcard`, an extension attribute.

EXAMPLE 6–1 Extension Query for `creditcard`

```
 /pp:PP/pp:Extension/ispp:PPISExtension[@name='creditcard']
Note: The prefix for the PPISExtension is different,
 and the schema for the PP extension is as follows:
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.sun.com/identity/liberty/pp"
  targetNamespace="http://www.sun.com/identity/liberty/pp">
  <xs:annotation>
      <xs:documentation>
      </xs:documentation>
  </xs:annotation>

  <xs:element name="PPISExtension">
     <xs:complexType>
        <xs:simpleContent>
           <xs:extension base="xs:string">
              <xs:attribute name="name" type="xs:string"
                use="required"/>
           </xs:extension>
        </xs:simpleContent>
     </xs:complexType>
   </xs:element>
</xs:schema>
```

Extension Attributes Namespace Prefix

The value of this attribute specifies the namespace prefix for the extensions defined in the "Extension Container Attributes" on page 128. This prefix is prepended to the element and helps to distinguish metadata from different XML schema namespaces.

Is ServiceUpdate Enabled

The SOAP Binding Service allows a service to indicate that requesters should contact it on a different endpoint or use a different security mechanism and credentials to access the requested resource. If selected, this attribute affirms that there is an update to the service instance.

Service Instance Update Class

The value of this attribute specifies the default implementation class `com.sun.identity.liberty.ws.idpp.plugin.IDPPServiceInstanceUpdate`. This class is used to update the information for the service instance.

Alternate Endpoint

The value of this attribute specifies an alternate SOAP endpoint to which a SOAP request can be sent.

Alternate Security Mechanisms

This attribute allows you to choose a security mechanism. For more information about this functionality and the mechanisms, see the *Liberty ID-WSF Security Mechanisms* specification.

Liberty Employee Profile Service

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

Note – Before implementing this sample, you must have two instances of Access Manager installed, running, and Liberty-enabled. Completing the steps in "`sample1` Directory" on page 209 will accomplish this.

The Liberty Employee Profile Service is a deployment of the ID-SIS-EP specification as discussed in "Liberty Employee Profile Service" on page 121. The `Readme.html` file in the sample directory provides detailed steps on how to deploy and configure this sample for use as a data service. See also Appendix A.

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 129
- "`com.sun.identity.liberty.ws.dst.service` Package" on page 130

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 6–1 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. For more information, see "`sis-ep` Directory" on page 211.

Developing A New Data Service

In addition to deploying an employee profile service, the Liberty Employee Profile Service sample can be used to deploy other custom data services that are based on the Liberty ID-WSF-DST. Sections 2 and 3 in the `Readme.html` file in the */AccessManager-base*`/SUNWam/samples/phase2/sis-ep` directory has detailed steps on how to deploy and configure data services. To use those instructions for a new data service, you need to write a new data service schema. This XML Schema Definition (XSD) document (as discussed in Appendix B) defines the service's data and data structure. After you write a new XSD file, use it to deploy your new data service instead of the `lib-id-sis-ep.xsd` file.

Note – Instructions on writing the XSD file are beyond the scope of this guide.

◆ ◆ ◆ CHAPTER 7

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.

This chapter covers the following topics:

Discovery Service Overview

The initial step in accessing identity data (as discussed in Chapter 6) is to determine where the information is located. For example, you must determine which identity service holds the principal's credit card information or which server stores the principal's calendar service. Typically, there are one or more services on a network that allow other entities to perform an action on identity data. Because clients are not expected to keep track of these services or to know which can be trusted, they require a *discovery service*. The *Liberty ID-WSF Discovery Service Specification* defines the framework that enables a client to locate the appropriate web service for retrieving, updating, or modifying a specific piece of identity data.

Note – For more information, see the *Liberty ID-WSF Discovery Service Specification*.

Discovery Service Concepts

A *discoverable web service* is assigned a service type unique resource identifier (URI) in the specification that defines it. This URI points to the Web Services Description Language (WSDL) file that describes the service's data, the operations that can be performed on it, and a protocol to perform the operations. The discoverable service specification itself adds the available ways the data can be exchanged. A *discovery service* is essentially a web service interface for discovery resources. A *discovery resource* is a registry of resource offerings. A *resource offering* defines an association between a piece of identity data and the service instance that provides access to that data. A *resource identifier* is a URI registered with the discovery service that points to a particular discovery resource.

When a client sends a request for some type of data, it includes a resource identifier that the Discovery Service uses to locate the web services provider (WSP) for the requested attributes. The Discovery Service returns a resource offering that contains the information necessary to locate the data.

Note – Because a provider hosting the Discovery Service can also be fulfilling other roles for an identity (such as a Policy Decision Point or an Authentication Authority), a query response also functions as a security token service. It provides a requester with the means of obtaining security tokens that can be used to invoke service instances returned.

Discovery Entries

One user account has one discovery resource. This discovery resource can include zero or more resource offerings. Storing resource offerings within a user profile supports both entry lookups and updates. Another option is to store discovery entries within a service, and assign that service to an organization or a role. This scenario supports only entry lookups using the discovery protocol although you can still update the entries using the console. For more information about discovery entries, see "Discovery Entries and Resource Offerings" on page 142.

XML Service Files

The Discovery Service is defined using the XML service file `amDisco.xml`. This file defines the attributes for the Discovery Service. All of the attributes in the Discovery Service can be managed through either the Access Manager Console or this file.

Note – For more information about service files, see the *Sun Java System Access Manager 7 2005Q4 Administration Guide*.

A second XML file, `amDisco_add.xml` is in `/AccessManager-base/SUNWam/upgrade/services50_sunIdentityServerDiscoveryService/10_20/data`. This file is used for upgrading Identity Server 6.2 to Access Manager 6.3. It lists the changes to the `amDisco.xml` file since the Identity Server release.

Discovery Service APIs

Access Manager contains several Java packages that are used by the Discovery Service. They include:

- "com.sun.identity.liberty.ws.disco Package" on page 135
- "com.sun.identity.liberty.ws.disco.plugins Package" on page 135
- "com.sun.identity.liberty.ws.interfaces Package" on page 135

Additional information is in "Discovery Service APIs" on page 152 and 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 "Common Service Interfaces" on page 197 in Chapter 10.

com.sun.identity.liberty.ws.disco Package

This package includes a client API that provides interfaces developers can use to communicate with the Discovery Service.

com.sun.identity.liberty.ws.disco.plugins Package

This package includes an interface that can be used to develop plug-ins. The package also contains some default plug-ins.

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

This package 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.

Discovery Service Architecture

The Access Manager Discovery Service includes Java and web services-based interfaces. Java applications use the client API (discussed in "Client APIs in com.sun.identity.liberty.ws.disco" on page 155) to form requests sent to the Discovery Service and to parse the responses received back from it. Requests are received by the Access Manager SOAP receiver, which constructs a SOAP message that incorporates the client request.

Note – The Access Manager SOAP Binding Service defines how to send and receive messages using SOAP, an XML-based messaging protocol. The SOAP receiver is a servlet that constructs the message using these definitions. For more information, see Chapter 8.

The SOAP message is then sent to the Discovery Service, which parses a discovery resource identifier from it. This identifier is used to find a matching user DN. The necessary information is then culled from the corresponding profile, a response is generated, and the response is sent back to the SOAP receiver. The SOAP receiver then sends the response back to the client. The following figure illustrates this architecture.

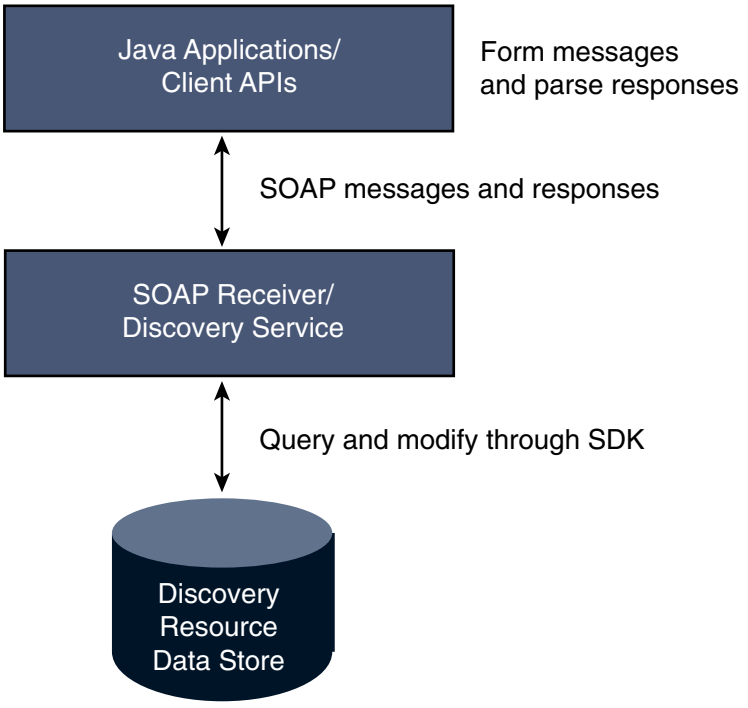

FIGURE 7–1 Discovery Service Architecture

Discovery Service Process

The following figure provides a high-level overview of the interaction between parties in a web services environment using the Discovery Service. In this scenario, the identity provider hosts the Discovery Service. The process is defined in more detail after the figure.

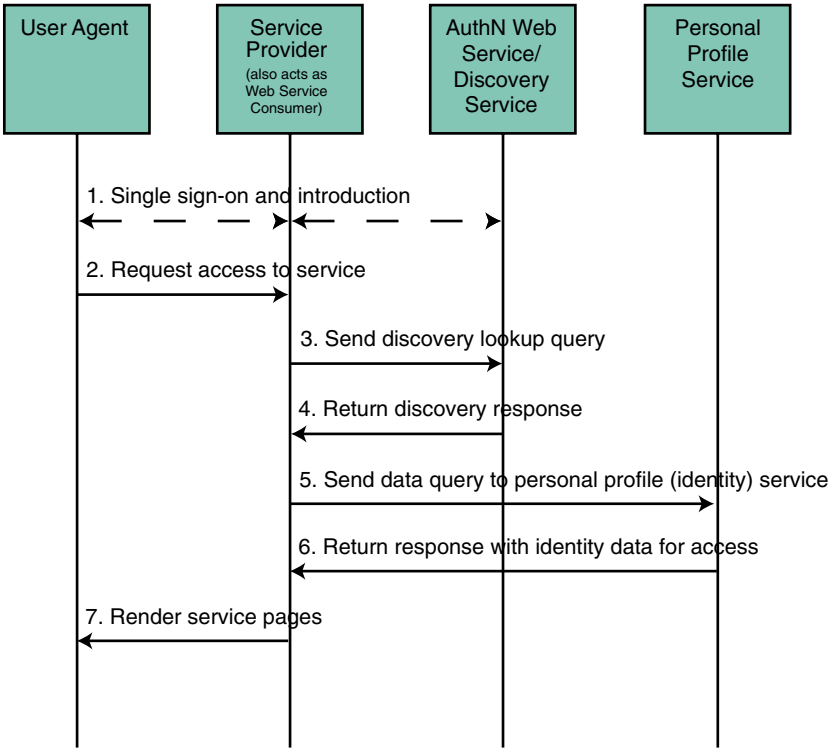

FIGURE 7–2 Participants and Process of the Discovery Service

1. The user logs in to a Liberty-enabled identity provider, is authenticated, and completes the *introduction* process, enabling single sign-on with other members of the authentication domain. More specifically, this is the process:
   a. Within a browser, the user types the URL for a Liberty-enabled service provider.
   b. The service provider collects the user's credentials and redirects the information to the identity provider for authentication.
   c. If the credentials are verified, the user is authenticated.
   d. Assuming the identity provider is the center of an authentication domain, that provider will notify the authenticated user of the option to federate any local identities created with member organizations. The user would then accept or decline this invitation to federate. By accepting the invitation, the user will be given the option of federation to a member organization's web site at each login. If the user accepts this option to federate, single sign-on is enabled.
2. After authentication, the user now requests access to services hosted by another service provider in the authentication domain.
3. The service provider sends a lookup query to the Discovery Service.

Information used by a client to contact Discovery Service is culled from the authentication statement.

4. The Discovery Service returns a discovery lookup response to the service provider.

   The lookup response contains the *resource offering* (defining an association between a piece of identity data and the service instance that provides access to it) for the user's Personal Profile Service.

5. The service provider then sends a query (using the "Data Services Template Specification" on page 40) to the Personal Profile Service instance.

   The required authentication mechanism specified in the Personal Profile Service resource offering must be followed.

6. The Personal Profile Service instance returns a Data Services Template response after collecting all required data.

   The Personal Profile Service authenticates and validates authorization, or policy, or both for the requested user and service provider. If user interaction is required for some attributes, the Interaction Service will be invoked to query the user for consents or for attribute values.

7. The service provider processes the Personal Profile Service response and renders HTML pages based on the original request and user authorization.

   A users' actual account information is not exchanged during federation. Thus, the identifier displayed on each provider site will be based on the local identity profile.

Discovery Service Attributes

The Discovery Service attributes are global attributes whose values are applied across the Access Manager configuration and inherited by every configured organization.

Note – For information about the types of attributes used in Access Manager, see the *Sun Java System Access Manager 7 2005Q4 Technical Overview*.

The Discovery Service attributes are:

- "Resource Offerings for Bootstrapping Resources" on page 141

Provider ID

This attribute takes a URI that points to the Discovery Service. Use the format *protocol*`://`*host*`:`*port*`/amserver/Liberty/disco`. This value can be changed only if other relevant attributes values are changed to match the new location.

Supported Authentication Mechanisms

This attribute specifies the authentication methods supported by the Discovery Service. These security mechanisms refer to the way a web service consumer authenticates to the web service provider or provides message-level security. By default, all available methods are selected. If an authentication method is not selected and a web services consumer (WSC) sends a request using that method, the request is rejected.

Supported Directives

This attribute allows you to specify a policy-related directive for a resource. If a service provider wants to use an unsupported directive, the request will fail. The following table describes the available options.

TABLE 7–1 Policy-Related Directives

Directive	Purpose
`AuthorizeRequester`	The Discovery Service should include a SAML assertion (containing an `AuthenticationStatement`) in its responses to enable the client to authenticate to the service instance hosting the resource.
`AuthenticateSessionContext`	The Discovery Service should include a SAML assertion (containing a `SessionContextStatement`) in its responses that indicate the status of the session.
`AuthorizeRequestor`	The Discovery Service should include a SAML assertion (containing a `ResourceAccessStatement`) in its responses that indicate whether the client is allowed to access the resource.
`EncryptResourceID`	The Discovery Service should encrypt the resource identifier in responses to all clients.

TABLE 7–1 Policy-Related Directives *(Continued)*

Directive	Purpose
`GenerateBearerToken`	For use with Bearer Token Authentication, the Discovery Service should generate a token that grants the bearer permission to access the resource.

Enable Policy Evaluation for `DiscoveryLookup`

If enabled, the service will perform a policy evaluation for the `DiscoveryLookup` operation. By default, the check box is not selected.

Enable Policy Evaluation for `DiscoveryUpdate`

If enabled, the service will perform a policy evaluation for the `DiscoveryUpdate` operation. By default, the check box is not selected.

`Authorizer` Plugin Class

The value of this attribute is the name and path to the class that implements the `com.sun.identity.liberty.ws.interfaces.Authorizer` interface used for policy evaluation of a WSC. The default class is `com.sun.identity.liberty.ws.disco.plugins.DefaultDiscoAuthorizer`.

Entry Handler Plugin Class

The value of this attribute is the name and path to the class that implements the `com.sun.identity.liberty.ws.disco.plugins.DiscoEntryHandler` interface. This interface is used to set or retrieve a principal's discovery entries. To handle discovery entries differently, implement the `com.sun.identity.liberty.ws.disco.plugins.DiscoEntryHandler` interface and set the implementing class as the value for this attribute. The default implementation for the Discovery Service is `com.sun.identity.liberty.ws.disco.plugins.UserDiscoEntryHandler`.

Classes For `ResourceIDMapper` Plugin

The value of this attribute is a list of classes that generate identifiers for a resource offering configured for an organization or role. `com.sun.identity.liberty.ws.interfaces.ResourceIDMapper` is an interface used to map a user identifier to the resource identifier associated with it. The Discovery Service provides two implementations for 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*

Different implementations may also be developed with the interface and added as a value of this attribute by clicking New and defining the following attributes:

- *Provider ID* takes as a value a URI that points to the Discovery Service. Use the format `http://`*host*`:`*port*`/amserver/Liberty/disco`. See "Provider ID" on page 139.
- *ID Mapper* takes as a value the class name and path of the implementing class.

Authenticate Response Message

If enabled, the service authenticates the response message. By default, the function is not enabled.

Generate `SessionContextStatement` for Bootstrapping

If enabled, this attribute specifies whether to generate a `SessionContextStatement` for bootstrapping. A `SessionContextStatement` conveys the session status of an entity. By default, this function is not enabled.

Encrypt NameIdentifier in Session Context for Bootstrapping

If enabled, the service encrypts the name identifier in a `SessionContextStatement`. By default, this function is not enabled.

Use Implied Resource; don't generate ResourceID for Bootstrapping

If enabled, the service does not generate a resource identifier for bootstrapping. By default, this function is not enabled.

Resource Offerings for Bootstrapping Resources

This attribute defines a resource offering for bootstrapping a service. After single sign-on (SSO), this resource offering and its associated credentials will be sent to the client in the SSO assertion. Only one resource offering is allowed for bootstrapping. By default, this offering contains information about the Discovery Service. For more information, see "Discovery Entries and Resource Offerings" on page 142.

The value of the Resource Offerings for Bootstrapping Resources attribute is a default value configured during installation. If you want to define a new resource offering, you must first delete the existing resource offering, then click New to define the attributes. If you want to edit an existing resource offering, click the name of the existing Service Type to modify the attributes.

Discovery Entries and Resource Offerings

In Access Manager, a discovery entry can be stored as a user attribute or as a dynamic attribute. When storing a discovery entry as a user attribute, one user account has one discovery resource that can include zero or more resource offerings. Storing resource offerings within a user profile supports both entry lookups and updates. When storing a discovery entry as a dynamic attribute, the entry can be assigned to a realm or a role. This scenario only supports entry lookups using the discovery protocol. More information is provided in the following sections:

- "Storing Discovery Entries as User Attributes" on page 142
- "Storing Discovery Entries as Dynamic Attributes" on page 145
- "Storing Discovery Entries for Bootstrapping" on page 150

Storing Discovery Entries as User Attributes

Discovery entries can be stored as a user attribute under a user's distinguished name (DN) using the Lightweight Directory Access Protocol (LDAP). Storing resource offerings within a user profile supports both entry lookups and updates. The following procedure explains how to access and create a user's resource offerings.

▼ To Access and Create a User's Resource Offerings

1. **In the Access Manager Console, click the Access Control tab.**
2. **Select the name of the realm that contains the user you want to modify.**
3. **Select Subjects to access user information.**
4. **Select the name of the user profile that you want to modify.**
5. **Select Services to access the user's services.**
6. **Click Add to configure the Discovery Service for this user.**
7. **Select Discovery Service and click Next.**
   The Discovery Service is added to the user's services.
8. **Select General to access the user's User Discovery Resource Offering attribute.**

9 Click Edit.

A User Discovery Resource Offering window opens.

10 Click Add in the User Discovery Resource Offering window.

11 (Optional) Type a value for the Resource ID Attribute.

This field defines an identifier for the resource offering.

12 Type the Resource ID Value.

This field defines the resource identifier. A *resource identifier* is a URI registered with the Discovery Service that point to a particular discovery resource. It is generated by the profile provider. The value of this attribute must not be a relative URI and should contain a domain name that is owned by the provider hosting the resource. If a discovery resource is exposed in multiple Resource Offerings, the Resource ID Value for all of those resource offerings would be the same. An example of a valid Resource ID value is `http://profile-provider.com/profiles/14m0B82k15csaUxs`.

Tip – `urn:libery:isf:implied-resource` can be used as a Resource ID Value when only one resource can be operated upon at the service instance being contacted. The URI only implicitly identifies the resource in question. In some circumstances, the use of this resource identifier can eliminate the need for contacting the discovery service to access the resource.

---

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

14 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 is `urn:liberty:id-sis-pp:2003-08`.

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

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. An example of a valid URI is `http://profile-provider.com`.

Note – The provider represented by the URI in the Provider ID attribute must also have a class entry in the `ResourceIDMapper` attribute. For more information, see "Classes For `ResourceIDMapper` Plugin" on page 140.

---

16 Click Add 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 of the SOAP-over-HTTP endpoint. The URI scheme must be HTTP or HTTPS as in `https://soap.profile-provider.com/soap`.

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.

17 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*.

18 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.

19 **Click OK.**

20 **Click Close to close the User Discovery Resource Offering window.**

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

Storing Discovery Entries as Dynamic Attributes

Due to the repetition inherent in storing discovery entries as user attributes, Access Manager has established the option of storing a discovery entry as a dynamic attribute within a role or a realm. The role or realm can then be assigned to an identity-related object, making the entry available to all users within the object. Unlike storing a discovery entry as a user attribute, this scenario only supports entry lookups, not updates.

Note – For more information about services, see the *Sun Java System Access Manager 7 2005Q4 Administration Guide*.

There are two ways in which you can store discovery entries as dynamic attributes. You can store them in a realm or in a role. The following sections describe the procedures:

- "To Store Discovery Entries as Dynamic Attributes in a Realm" on page 145
- "To Store Discovery Entries as Dynamic Attributes in a Role" on page 147

▼ To Store Discovery Entries as Dynamic Attributes in a Realm

To create a discovery entry as a dynamic attribute in a realm, the Discovery Service must first be added and a template created.

1 **In the Access Manager Console, click the Access Control tab.**

2 **Select the name of the realm you want to modify.**

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

4 **Click Add to add the Discovery Service to the realm.**

A list of available services is displayed.

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

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

6 **Select Subjects to access user information.**

7 **Select the name of the user you want to modify.**

8 **Select Services to add the Discovery Service to the user.**

9 Click Add to add the Discovery Service to the user.

A list of available services is displayed.

10 Select Discovery Service and click Next to add the service.

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

11 Using the path displayed on top of the Access Manager Console, click Edit Realm.

12 Click Services to access the realm's services.

13 Select Discovery Service to add a resource offering to the service.

14 Click Add.

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

16 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 is `urn:liberty:id-sis-pp:2003-08`.

17 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. An example of a valid URI is `http://profile-provider.com`.

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 "Classes For `ResourceIDMapper` Plugin" on page 140.

---

18 Click Add 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 of the SOAP-over-HTTP endpoint. The URI scheme must be HTTP or HTTPS as in `https://soap.profile-provider.com/soap`.

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.

19 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*.

20 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.

21 Click OK.

22 Click Close to close the Discovery Resource Offering window.

23 Click Save to save the configuration.

▼ To Store Discovery Entries as Dynamic Attributes in a Role

To create a discovery entry as a dynamic attribute in a role, the Discovery Service must first be added and a template created.

1 In the Access Manager Console, click the Access Control tab.

2 Select the name of the realm you want to modify.

3. **Select Subjects to access the realm's identity information.**

4. **Select Role to access the realm's role information.**

5. **Select the name of the role you want to modify.**

   Alternately, you can create a new role and then select the name of this new role.

6. **Under Services, click Add to add the Discovery Service to the role.**

   A list of available services is displayed.

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

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

8. **Select Discovery Service to add a resource offering to the service.**

9. **Click Add.**

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

11. **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 is `urn:liberty:id-sis-pp:2003-08`.

12. **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. An example of a valid URI is `http://profile-provider.com`.

    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 "Classes For `ResourceIDMapper` Plugin" on page 140.

    ---

13. **Click Add 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 of the SOAP-over-HTTP endpoint. The URI scheme must be HTTP or HTTPS as in `https://soap.profile-provider.com/soap`.

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.

14 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*.

15 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.

16 Click OK.

17 Click Close to close the Discovery Resource Offering window.

18 Click Save to save the configuration.

Storing Discovery Entries for Bootstrapping

Before a WSC can contact the Discovery Service for a resource offering, the WSC needs to find the Discovery Service. Thus, an initial resource offering for locating the Discovery Service is sent back to the WSC in a single sign-on assertion. The following procedure describes how to configure a global attribute for bootstrapping the Discovery Service. Unlike storing a discovery entry as a user attribute, this scenario only supports entry lookups, not updates.

▼ To Store Discovery Entries for Bootstrapping

1 In the Access Manager Console, select the Web Services tab.

2 Under Web Services, click the Discovery Service tab.

3 Choose New under the Resource Offerings for Bootstrapping Resources attribute.

By default, the resource offering for bootstrapping the Discovery Service is already configured. In order to create a new resource offering, you must first delete the default resource offering.

4 (Optional) Type a description of the resource offering.

5 Enter a URI for the value of the Service Type attribute.

This field 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 is `urn:liberty:disco:2003-08`.

6 Enter a URI for the value of the Provider ID attribute.

This attribute contains the URI of the provider of the service instance. This is useful for resolving trust metadata needed to invoke the service instance. A single physical provider may have multiple provider IDs. An example of a valid URI is `http://sample_disco.com`.

Note – The provider represented by the URI in the Provider ID attribute must also have an entry in the Classes for ResourceIDMapper Plugin attribute. For more information, see "Classes For `ResourceIDMapper` Plugin" on page 140.

---

7 Click Add Description to define a security mechanism ID.

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 you wish to add and click the Add button. To arrange the priority of the list, select the mechanism and use the Move Up or Move Down buttons.

b. Type a value for the End Point URL.

This value is the URL of the SOAP-over-HTTP endpoint. The URI scheme must be HTTP or HTTPS as in `https://soap.profile-provider.com/soap`.

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

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

d. Click OK to save the configuration.

8 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*.

9 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.

10 Click OK to complete the configuration.

Discovery Service APIs

By default, a discovery service is implemented as one of the identity web services in Access Manager. The Discovery Service APIs provide the following implementations and interfaces:

`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.

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 2005Q4 Administration Guide*.

Policy definitions for the Discovery Service are configured using the Access Manager Console. The procedure is as follows.

▼ To Configure 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 "Common Service Interfaces" on page 197 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 2005Q4 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 "Classes For `ResourceIDMapper` Plugin" on page 140 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 "Common Service Interfaces" on page 197 and the Java API Reference in */AccessManager-base*`/SUNWam/docs` or on docs.sun.com.

`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 "Entry Handler Plugin Class" on page 140. The default implementations of this interface are described in the following table.

TABLE 7–2 Implementations of `com.sun.identity.liberty.ws.disco.plugins.DiscoEntryHandler`

Class	Description
`UserDiscoEntryHandler`	Gets or modifies discovery entries stored in the user's entry as a value of the `sunIdentityServerDiscoEntries` attribute. The `UserDiscoEntryHandler` implementation is used in business-to-consumer scenarios such as the Liberty Personal Profile Service.

TABLE 7–2 Implementations of `com.sun.identity.liberty.ws.disco.plugins.DiscoEntryHandler` *(Continued)*

Class	Description
`DynamicDiscoEntryHandler`	Gets discovery entries stored as a value of the `sunIdentityServerDynamicDiscoEntries` dynamic attribute in the Discovery Service. Modification of these entries is not supported and always returns `false`. The resource offering is saved in an organization or a role. The `DynamicDiscoEntryHandler` implementation is used in business-to-business scenarios such as the Liberty Employee Profile service.
`UserDynamicDiscoEntryHandler`	Gets a union of the discovery entries stored in the user entry `sunIdentityServerDiscoEntries` attribute and discovery entries stored in the Discovery Service `sunIdentityServerDynamicDiscoEntries` attribute. It modifies only discovery entries stored in the user entry. The `UserDynamicDiscoEntryHandler` implementation can be used in both business-to-consumer and business-to-business scenarios.

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 7–3 Discovery Service Client APIs

Class	Description
`Description`	Represents a DescriptionType element of a service instance.
`Directive`	Represents a discovery service DirectiveType element.
`DiscoveryClient`	Provides methods to send Discovery Service queries and modifications.
`EncryptedResourceID`	Represents an EncryptionResourceID element for the Discovery Service.
`InsertEntry`	Represents an Insert Entry for Discovery Modify request.
`Modify`	Represents a discovery modify request.
`ModifyResponse`	Represents a discovery response to a modify request.

TABLE 7–3 Discovery Service Client APIs *(Continued)*

Class	Description
`Query`	Represents a discovery Query object.
`QueryResponse`	Represents a response to a discovery query request.
`RemoveEntry`	Represents a remove entry element for the discovery modify request.
`RequestedService`	Enables the requester to specify that all the resource offerings returned must be offered through a service instance that complys with one of the specified service types.
`ResourceID`	Represents a Discovery Service Resource ID.
`ResourceOffering`	Associates a resource with a service instance that provides access to that resource.
`ServiceInstance`	Describes a web service at a distinct protocol endpoint.

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. For more information, see Appendix A.

◆ ◆ ◆ CHAPTER 8

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.

This chapter covers the following topics:

SOAP Binding Service Overview

The Liberty Identity Web Services Framework (Liberty ID-WSF) and Liberty Identity Service Interface Specifications (Liberty ID-SIS) components of the Liberty Alliance Project specifications use messages to convey identity data between providers. Access Manager has implemented the *Liberty ID-WSF SOAP Binding Specification* (Liberty ID-WSF-SBS) as the method of transport for this purpose. The specification defines SOAP as the binding to the Hypertext Transport Protocol (HTTP), which is itself layered onto the TCP/IP stack.

Note – For more information, see the *Liberty ID-WSF SOAP Binding Specification*.

XML Service File

The Access Manager SOAP Binding Service is defined using the XML service file `amSOAPBinding.xml`. This file defines the attributes for the SOAP Binding Service which can be managed through the Access Manager Console or the XML file.

Note – For more information on service files, see the *Sun Java System Access Manager 7 2005Q4 Administration Guide*.

The Liberty ID-WSF-SBS also defines an XML schema for use in building the SOAP messages. This XML Schema Definition (XSD) file is on the Liberty Alliance Project web site. Version 1.0 is also reproduced in Appendix B.

SOAP Binding Service APIs

The Access Manager SOAP Binding Service includes a Java package named `com.sun.identity.liberty.ws.soapbinding`. For more information about these interfaces, see "SOAP Binding Service Package" on page 161.

SOAP Binding Process

In the SOAP Binding process, an identity service calls the client-side application programming interface (API) to construct a message and send it to the SOAP endpoint URL. The URL is, in effect, a servlet that receives and processes SOAP messages.

Note – The Discovery Service, implemented Data Services Template services (including the Liberty Personal Profile Service and the sample Employee Profile Service), and the Authentication Web Service use the SOAP Binding Service client API.

The SOAP Receiver servlet receives the message, verifies the signature, and constructs a second message. The SOAP Receiver servlet then invokes the correct request handler class to send this second message to the corresponding service for a response.

Note – `com.sun.identity.liberty.ws.soapbinding.RequestHandler` is an interface that must be implemented on the server side by any Liberty-based web service using the SOAP Binding Service. For more information, see "Request Handler List" on page 159.

The service processes the second message, generates a response, and sends that response back to the SOAP Receiver servlet. The SOAP receiver, in turn, sends the response back to the service for processing.

Note – Before invoking a corresponding service, the SOAP framework might also do the following:

- Authenticate the sender identity to verify the credentials of a WSC peer, probably by verifying its client certificate.
- Authenticate the invoking identity to verify the credentials of a WSC on behalf of a user to verify whether the user has been authenticated. This depends on the security authentication profile.
- Granular authorization to authorize the WSC before processing a service request.

SOAP Binding Service Attributes

The SOAP Binding Service attributes are *global* attributes. The values of these attributes are carried across the Access Manager configuration and inherited by every organization.

Note – For information about the types of attributes used in Access Manager, see the *Sun Java System Access Manager 7 2005Q4 Technical Overview*.

Attributes for the SOAP Binding Service are defined in the `amSOAPBinding.xml` service file. The SOAP Binding Service attributes are as follows:

Request Handler List

The Request Handler List stores information about the classes implemented from the `com.sun.identity.liberty.ws.soapbinding.RequestHandler` interface. The SOAP Binding Service provides the interface to process requests and return responses. It must be implemented on the server side for each Liberty-based web service that uses the SOAP Binding Service.

Note – The Discovery Service, implemented Data Services Template (DST) services (including the Liberty Personal Profile Service and the sample Employee Profile Service, if deployed), and the Authentication Web Service use the SOAP Binding Service client API.

To add a new implementation, click New and define values for the following parameters.

Key Parameter

The Key parameter is the last part of the URI path to a SOAP endpoint. The SOAP endpoint in Access Manager is the SOAP Receiver servlet. The URI to the SOAP Receiver uses the format `protocol://`*host*`:`*port*`/`*deloy-uri*`/Liberty/`*key*. If you define `disco` as the Key, the URI path to the SOAP endpoint for the corresponding Discovery Service would be *protocol*`://`*host*`:`*port*`/amserver/Liberty/disco`.

Note – Different service clients use different keys when connecting to the SOAP Receiver.

`Class` Parameter

The `Class` parameter specifies the name of the class implemented from `com.sun.identity.liberty.ws.soapbinding.RequestHandler` for the particular web service. For example, `class=com.example.identity.liberty.ws.disco.DiscoveryService`.

`SOAP Action` Parameter

The optional `SOAP Action` can be used to indicate the intent of the SOAP HTTP request. The SOAP processor on the receiving system can use this information to determine the ultimate destination for the service. The value is a URI. No defined value indicates no intent.

Note – SOAP places no restrictions on the format or specificity of the URI or that it is resolvable.

Web Service Authenticator

This attribute takes as a value the implementation class for the Web Service Authenticator interface. This class authenticates a request and generates a credential for a WSC.

Note – This interface is not public. The value of the attribute is configured during installation.

Supported Authentication Mechanisms

This attribute specifies the authentication mechanisms supported by the SOAP Receiver. Authentication mechanisms offer user authentication as well as data integrity and encryption. By default, all available authentication mechanisms are selected. If a mechanism is not selected and a WSC sends a request using it, the request is rejected. Following is a list of the supported authentication mechanisms:

- `urn:liberty:security:2003-08:null:null`
- `urn:liberty:security:2003-08:null:X509`
- `urn:liberty:security:2003-08:null:SAML`
- `urn:liberty:security:2004-04:null:Bearer`
- `urn:liberty:security:2003-08:TLS:null`
- `urn:liberty:security:2003-08:TLS:X509`
- `urn:liberty:security:2003-08:TLS:SAML`
- `urn:liberty:security:2004-04:TLS:Bearer`
- `urn:liberty:security:2003-08:ClientTLS:null`
- `urn:liberty:security:2003-08:ClientTLS:X509`

- `urn:liberty:security:2003-08:ClientTLS:SAML`
- `urn:liberty:security:2004-04:ClientTLS:Bearer`

Note – For more complete information about authentication mechanisms and their level of security, see the *Liberty ID-WSF Security Mechanisms* specification.

SOAP Binding Service Package

The Access Manager SOAP Binding Service includes a Java package named `com.sun.identity.liberty.ws.soapbinding`. This package provides classes to construct SOAP requests and responses and to change the contact point for the SOAP binding. The following table describes some of the available classes. For more detailed information, see the Java API Reference in */AccessManager-base*`/SUNWam/docs` or on docs.sun.com.

TABLE 8–1 SOAP Binding Service Classes

Class	Description
`Client`	Provides a WSC with a method to send requests using a SOAP connection with a WSP.
`ConsentHeader`	Defines the SOAP element named `Consent`.
`CorrelationHeader`	Defines the SOAP element named `Correlation`.
`ProcessingContextHeader`	Defines the SOAP element named `ProcessingContext`.
`ProviderHeader`	Defines the SOAP element named `Provider`.
`RequestHandler`	Defines an interface that needs to be implemented by each web service in order to receive a request from your web service client. After implementing the handler class, the user must register the class in the SOAP Binding Service so the SOAP layer knows where to forward incoming WSC requests.
`Message`	Used by both the web service client and server to construct SOAP requests and responses.
`ServiceInstanceUpdateHeader`	Allows a service to change the endpoint on which requesters will contact it.
`ServiceInstanceUpdateHeader.Credential`	Allows a service to use a different security mechanism and credentials to access the requested resource.
`SOAPBindingException`	Represents an error that has occurred while processing a SOAP request and response.
`SOAPFault`	Defines the SOAP element named `Fault`.

TABLE 8–1 SOAP Binding Service Classes *(Continued)*

Class	Description
`SOAPFaultDetail`	Defines the SOAP element named `Consent`.
`SOAPFaultException`	Represents a SOAP fault while processing a SOAP request.
`UsageDirectiveHeader`	Defines the SOAP element named `UsageDirective`.

See Appendix A for sample code and files to help you understand the implementation of the Liberty Alliance Project specifications.

See "PAOS Binding" on page 203 for information on this reverse HTTP binding for SOAP.

PART IV

SAML Administration and Application Programming Interfaces

◆ ◆ ◆ CHAPTER 9

SAML Administration

Sun Java™ System Access Manager uses the Security Assertion Markup Language (SAML) as the means for exchanging security information. SAML uses an eXtensible Markup Language (XML) framework to achieve interoperability between vendor platforms that provide SAML assertions. This chapter explains SAML and defines how it is used within Access Manager.

This chapter covers the following topics:

SAML Overview

SAML is an open-standard protocol that defines user authentication, entitlements, and attribute information in XML documents. The Organization for the Advancement of Structured Information Standards (OASIS) drives the development of SAML 1.0 and 1.1, the versions supported in Access Manager 7 2005Q4.

Note – For information and specifications, see the OASIS Security Services (SAML) Technical Committee web site.

The SAML documents can be used to exchange security information between an *authority* and a *trusted* partner site. The security information that is exchanged deals with a subject's authentication status, access authorization, and attribute information. A *subject* is an entity in a particular domain. A person identified by an email address is a subject, as might be a printer. A *SAML authority*, sometimes called the *asserting party*, is a platform or application that has been integrated with the SAML API, allowing it to relay security information. *Trusted* partner sites receive the security information and rely on its authenticity.

Note – All domains need to form a trust relationship before they can share information about a subject's identity. How this is accomplished is beyond the scope of this guide.

Comparison of SAML and Liberty Specifications

SAML was designed by vendors to address the issue of cross-domain single sign-on. The Liberty Alliance Project was formed to develop technical specifications that would solve business process problems. These issues include single sign-on, but also incorporate protocols for account linking and consent, among others. SAML, on the other hand, does not solve issues such as privacy, single logout, and federation termination.

The SAML 1.0 and 1.1 specifications and the Liberty Alliance Project specifications do not compete with one another. They are complementary. In fact, the Liberty Alliance Project specifications leverage profiles from the SAML specifications. The decision of whether to use SAML or the Liberty specifications depends on your goal. In general, SAML should suffice for single sign-on basics. The Liberty Alliance Project specifications can be used for more sophisticated functions and capabilities, such as global sign-out, attribute sharing, web services. The following table lists the benefits of the two.

TABLE 9–1 Benefits of the SAML and the Liberty Alliance Project Specifications

SAML Uses	Liberty Alliance Project Uses
Cross-domain single sign-on	Single sign-on *only* after user federation
No user federation	User federation
No privacy control, best for use within one company	Built on top of SAML
User identifier is sent in plain text	User identifier is sent as a unique handle

Note – SAML Version 2.0 has been integrated into the Liberty Alliance Project specifications. This version is planned for implementation in an upcoming release of Access Manager.

SAML Architecture in Access Manager

SAML security information is expressed in the form of an assertion about a subject. An *assertion* is a package of verified security information that supplies one or more statements concerning a subject's authentication status, access authorization decisions, or identity attributes. Assertions are issued by the SAML authority, and received by partner sites defined by the authority as *trusted*. SAML authorities use different sources to configure the assertion information, including external data stores or assertions that have already been received and verified. The following figure illustrates how SAML interacts with the other components in Access Manager.

Note – Although Federation (as described in Chapter 3) integrates aspects of the SAML specifications, its usage of SAML is independent of the SAML component as described in this chapter.

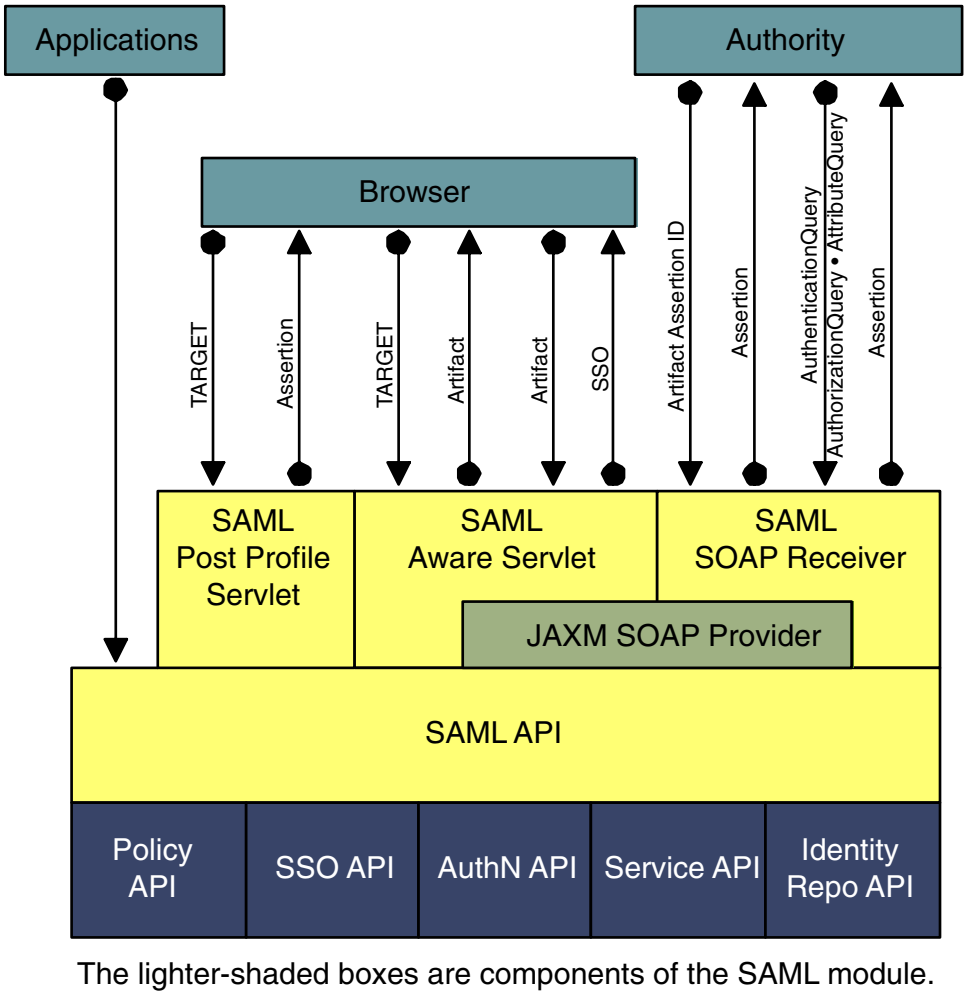

FIGURE 9–1 SAML Interaction in Access Manager

SAML allows Access Manager to work in the following ways:

- Users can authenticate using Access Manager and access trusted partner sites without having to reauthenticate.

Note – This single sign-on process is independent of the proprietary Access Manager process discussed in the *Sun Java System Access Manager 7 2005Q4 Administration Guide*.

- Access Manager acts as a policy decision point, allowing external applications to access user authorization information for the purpose of granting or denying access to their resources. For example, employees of an organization can be allowed to order office supplies from suppliers if they are authorized to do so.
- Access Manager acts as both an attribute authority (allowing trusted partner sites to query a subject's attributes) and an authentication authority (allowing trusted partner sites to query a subject's authentication information).
- Two parties in different security domains can validate each other for the purpose of performing business transactions.
- The SAML API can be used to build Authentication, Authorization Decision, and Attribute Assertions.
- The SAML service permits an XML-based digital signature signing and verifying functionality to be plugged into it.

Using SAML

The SAML can be accessed using a web browser or the SAML API. An end user authenticates to Access Manager using a web browser and, once authorized to do so, accesses URLs from trusted partner sites. Developers integrate the API into their applications to exchange security information with Access Manager. For example, a Java application can use the SAML API to achieve single sign-on. After obtaining a `SSOToken` from Access Manager, the application can call the `doWebArtifact()` method of the `SAMLClient` class, which will send a SOAP request for authorization information to Access Manager and, if applicable, redirect the application to the destination site. For more information, see "SAML API" on page 188.

Elements of SAML

The following sections describe the elements of the SAML component:

Assertion Types

SAML assertions are a declaration of facts about a principal. For example, an assertion can be made that a particular client was granted update privileges to a specific database resource at a certain time. Assertions are constructed in XML based on the SAML assertion schema. Assertions are built from

the user's session information and optional attribute information using the `siteAttributeMapper` class. For more information, see "`SiteAttributeMapper` and `PartnerSiteAttributeMapper` Interfaces" on page 190.

Note – One assertion can contain many different statements made by the authority.

The SAML specification provides for different types of assertions:

- An *authentication assertion* declares that the specified subject has been authenticated by a particular means at a particular time. This information is declared in an `AuthenticationStatement` element. In Access Manager, the Authentication Service is the authentication authority. The following code example illustrates a sample authentication assertion.

```
<?xml version="1.0" encoding="UTF-8" ?>
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
 MajorVersion="1" MinorVersion="0" AssertionID="random-182726"
 Issuer="sunserver.example.com" IssueInstant="2001-11-05T17:23:00GMT-02:00">
        <saml:AuthenticationStatement
         AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
         AuthenticationInstant="2001-11-05T17:22:00GMT-02:00">
            <saml:Subject>
                <saml:NameIdentifier
                 NameQualifier="example.com">John Doe
                </saml:NameIdentifier>
            </saml:Subject>
        </saml:AuthenticationStatement>
</saml:Assertion>
```

- An *attribute assertion* declares that the specified subject is associated with the specified attribute. This information is declared in an `AttributeStatement` element. The identity data store that is networked with Access Manager is the attribute authority.
- An *authorization decision assertion* declares that the specified subject's request for access to a specified resource has been granted or denied. This information is declared in an `AuthorizationDecisionStatement` element. In Access Manager, the Policy Service is the authorization authority.

Profile Types

A *profile* is a set of rules that defines how to embed and extract SAML assertions. The profile describes how the assertions are combined with other objects by an authority, transported from the authority, and subsequently processed at the trusted partner site. Access Manager supports two profiles: the Web Browser Artifact Profile and the Web Browser POST Profile. Both profiles use HTTP. Either can be used in single sign-on between two SAML-enabled entities, allowing an authenticated user to access resources from a trusted partner site. Each profile has its benefits:

- The Web Browser Artifact Profile requires less processing overhead because there is no assertion signing as there is in the Web Browser POST Profile.
- The Web Browser Artifact Profile works without browsers enabled with JavaScript technology. It is considered more secure than the Web Browser POST Profile.
- The Web Browser POST Profile does not require SOAP. This profile is more firewall-friendly and involves fewer steps and less server-side processing.

The profile methods can be initiated through a web browser or the SAML API. For more information about the API method, see "SAML API" on page 188.

Web Browser Artifact Profile

The Web Browser Artifact Profile defines interaction between three parties: a user equipped with a web browser, an authority site, and a trusted partner site. The SOAP communication should be either Basic Authentication or Client Certificate Authentication over SSL. Note that XML signing is a stronger alternative.

1. When an authenticated user attempts to access a trusted partner (generally by clicking a link), the user is directed to a transfer service at the authority site.

   In Access Manager, the transfer service is SAMLAwareServlet. The base of the transfer service URL is `http(s)://`*access-manager-host.domain*`:`*port*`/`*deploy-uri*`/SAMLAwareServlet`. The URL is appended with the location to which the user is requesting access (`?TARGET=`*URL-of-destination*).

2. SAMLAwareServlet receives the information and compares the SAML module's list of Trusted Partners against the user's `TARGET` location.

   Only targets that are configured in the Trusted Partners attribute of the SAML module are accessible. For more information about this attribute, see "Trusted Partners" on page 183.

3. Assuming the `TARGET` location was found in the list of Trusted Partners, SAMLAwareServlet looks for and validates the session token from the inbound request.

   Without a valid session token, Access Manager will not create an assertion.

4. Assuming a valid session token, SAMLAwareServlet creates an artifact and a corresponding assertion.

   An *artifact* is carried as part of the URL and points to an assertion and its source. An artifact is not (and does not contain) security information. The assertion contains the security information. For more information, see "`SiteAttributeMapper` and `PartnerSiteAttributeMapper` Interfaces" on page 190.

   Note – The need to send an artifact rather than the assertion itself is dictated by the restrictions on URL size that are imposed by many web browsers.

   ---

5. SAMLAwareServlet redirects the user's browser to the Artifact Receiver URL with a query string that contains the artifact and the original `TARGET` location.

Note – In Access Manager, the Artifact Receiver URL and SAMLAwareServlet are the same. Other SAML implementations might not integrate the two functions.

6. At the Artifact Receiver URL, the artifact is extracted from the query string to locate the SOAP Receiver URL at the trusted partner site.

   The SAML API extracts the source ID from the artifact and uses it to locate the SOAP Receiver URL at the trusted partner site. For more information about the use of SOAP, see "SAML SOAP Receiver" on page 175.

7. A SOAP query that contains the artifact is sent to the SOAP Receiver URL at the trusted partner site that is requesting the assertion to which the artifact points.
8. The SOAP Receiver URL accepts the returned artifact query from the trusted partner site and responds by sending the correct assertion in a SOAP response.
9. The assertion is processed, mapping the user account information from the trusted partner site to the target site's user account.

   The user is either granted or denied access to the trusted partner site. If access is granted, a `SSOToken` is generated, a cookie is set to the browser, and the user is redirected to the `TARGET` location.

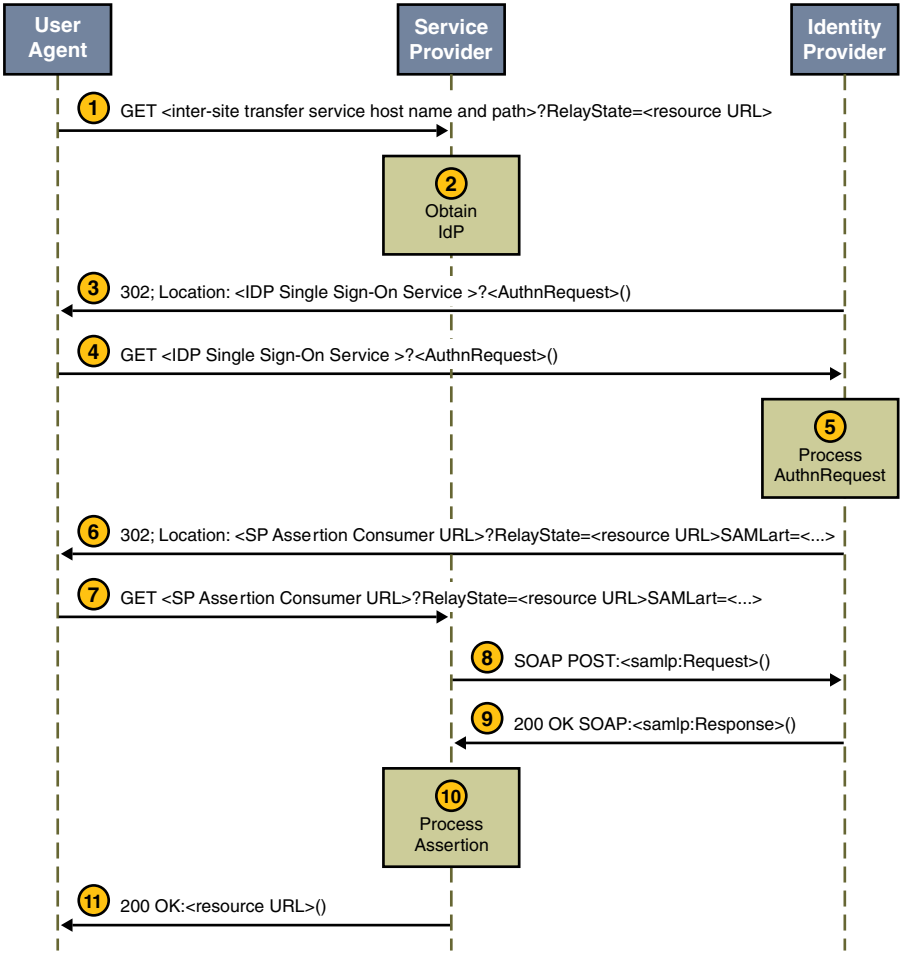

FIGURE 9–2 Web Browser Artifact Profile Interactions

A sample has been provided to test the Web Browser Artifact Profile function. See "SAML Samples" on page 193 for more information.

Web Browser POST Profile

The Web Browser POST Profile allows security information to be supplied to a trusted partner site using the HTTP POST method (without the use of an artifact). This interaction consists of two parts. The first part is between a user with a web browser and Access Manager. The second part is between the same user and the trusted partner site. The content of the POST should be signed to ensure message integrity, and the method of transport should be SSL.

Note – The POST profile function is provided by either of two means: an HTTP request using SAMLPOSTProfileServlet, or an `SAMLClient` API call [`doWebPost()`] to a Java application.

- The first interaction of the Web Browser POST Profile is as follows:
  1. An authenticated user attempts to access a trusted partner site using a web browser (usually by clicking a link), and the user is redirected to a transfer service at the authority site.

     In Access Manager, the transfer service is SAMLPostProfileServlet. The base of the transfer service URL is `http(s)://`*access-manager-host.domain*`:`*port*`/`*deploy-uri*`/SAMLPOSTProfileServlet`. This URL is appended with the location to which the user is requesting access (`?TARGET=`*URL-of-destination*).

     Note – SAMLPostProfileServlet provides functions for both Web Browser POST Profile interactions.

  2. Access Manager obtains the `TARGET` location from the request and matches it against the trusted partners configured in the Trusted Partners attribute of the SAML module.

     For more information, see "Trusted Partners" on page 183.

  3. Access Manager generates an assertion using the `AssertionManager` class of the SAML API.

     For information about the `AssertionManager` class, see "`com.sun.identity.saml` Package" on page 188.

  4. Access Manager forms, signs, and Base64 encodes a `SAMLResponse` that contains the assertion.
  5. Access Manager generates an HTML form that contains both the `SAMLResponse` and the `TARGET` as parameters and posts the form as an HTTP response back to the user's browser.
  6. The user's browser is then directed to the location based on this information.
- The second interaction of the Web Browser POST Profile is as follows:
  1. The trusted partner site obtains the `TARGET` and `SAMLResponse` from the redirected request.
  2. The trusted partner site decodes the `SAMLResponse`, verifies the signature on the `SAMLResponse`, and obtains and verifies the SAML response.

     The trusted partner site also verifies the assertion inside the `SAMLResponse` and enforces single sign-on policy.

  3. Assuming a positive authentication, the trusted partner site obtains or creates an `SSOToken` and redirects the authenticated user to the `TARGET` location.

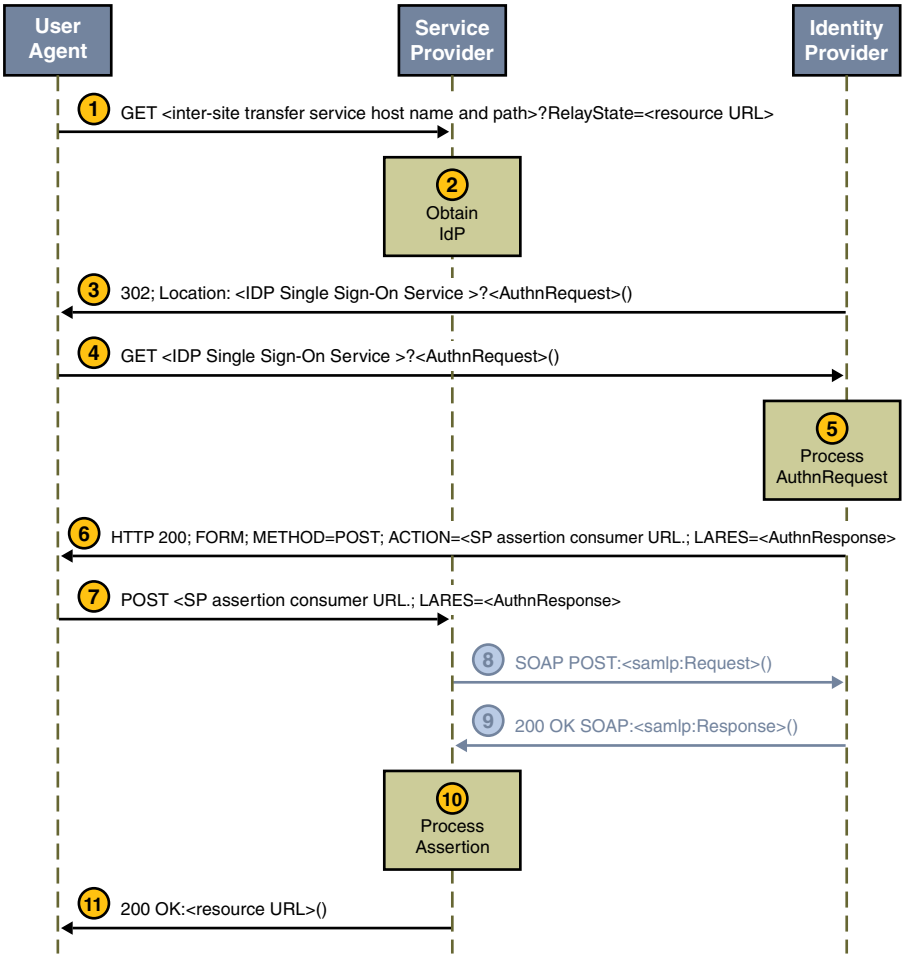

FIGURE 9–3 Web Browser POST Profile Interactions

A sample has been provided to test the Web Browser POST Profile function. See "SAML Samples" on page 193.

Single-Use Policy With POST Profile

According to the SAML specifications, the trusted partner site *must* ensure a single-use policy for SSO assertions that are communicated using the Web Browser POST Profile. SAMLPOSTProfileServlet maintains a store of SSO assertion identifiers and the time that they expire. When an assertion is received, the servlet first checks for an entry in the map. If an entry exists, the servlet returns an error. If an entry does not exist, the assertion identifier and expiration time are saved to the map. `POSTCleanUpThread` removes expired assertion identifiers periodically.

SAML SOAP Receiver

Assertions are exchanged between Access Manager and inquiring parties using the <Request> and <Response> XML constructs defined in the SAML specification. These SAML constructs are then integrated into SOAP messages for transport.

Note – A SAML <Request> can contain queries for authentication status, authorization decisions, attribute information, and one or more assertion identifier references or artifacts.

Access Manager uses SOAP, a message communications specification that integrates XML and HTTPS, to transport the SAML constructs. The request is received by SAML SOAP Receiver, a servlet that receives a SOAP message, extracts the SAML request, and responds with another SOAP message that contains the requested assertion. SAML SOAP Receiver responds to queries for authentication, attributes, or authorization decisions (including those that have an artifact) by returning assertions. The access URL for SAML SOAP Receiver is `http(s)://`*access-manager-host.domain*`:`*port*`/`*deploy-uri*`/SAMLSOAPReceiver`.

Note – SAML SOAP Receiver only supports the POST method.

SOAP Messages

SOAP messages consist of three parts: an envelope, header data, and a message body. The SAML <Request> and <Response> elements are enclosed in the message body. A client transmits a SAML <Request> element within the body of a SOAP message to an entity.

Note – The SAML API and the Java API for XML Messaging (JAXM) are used to construct SOAP messages and send them to SAML SOAP Receiver.

The following two samples illustrate a SOAP exchange for the "Web Browser Artifact Profile" on page 170. The first is a request for an authentication assertion.

EXAMPLE 9–1 SOAP Request for Authentication Assertion Using Web Browser Artifact Profile

```
POST /authn HTTP/1.1
Host: idp.example.com
Content-type: text/xml
Content-length: nnnn
<soap-env:Envelope
xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
<soap-env:Header/>
<soap-env:Body>
<samlp:Request xmlns="urn:oasis:names:tc:SAML:1.0:protocol"
    xmlns:lib="http://projectliberty.org/schemas/core/2002/12"
    xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
    xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
```

EXAMPLE 9–1 SOAP Request for Authentication Assertion Using Web Browser Artifact Profile *(Continued)*

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
IssueInstant="2002-12-12T10:08:56Z"
MajorVersion="1"
MinorVersion="0"
RequestID="e4d71c43-c89a-426b-853e-a2b0c14a5ed8"
id="ericssonb6dc3636-f2ad-42d1-9427-220f2cf70ec1"
xsi:type="lib:SignedSAMLRequestType">
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
        <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
        </ds:CanonicalizationMethod>
        <ds:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
        </ds:SignatureMethod>
        <ds:Reference URI="#ericssonb6dc3636-f2ad-42d1-9427-220f2cf70ec1">
            <ds:Transforms>
                <ds:Transform
                    Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature">
                </ds:Transform>
                <ds:Transform
                    Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                </ds:Transform>
            </ds:Transforms>
            <ds:DigestMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
            </ds:DigestMethod>
            <ds:DigestValue>+k6TnolGkIPKZlpUQVyok8dwkuE=</ds:DigestValue>
        </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>
        wXJMVoPO1V1jFnWJPyOWqP5Gqm8A1+/2b5gNzF4L4LMu4yEcRtttLdPPT3bvhwkwHXjL9
        NuOFumQ5YEyiVzlNcjAxX0LfgwutvEdJb748IU4L+8obXPXfqTZLiBK1RbHCRmRvjlPIu
        22oGCV6EwuiWRvOD6Ox9svtSgFJ+iXkZQ
    </ds:SignatureValue>
    <ds:KeyInfo>
        <ds:X509Data>
        <ds:X509Certificate>
        MIIDMTCCApqgAwIBAgIBHDANBgkqhkiG9w0BAQQFADCBlTELMAkGA1UEBhMCVVMxCzAJB
        gNVBAcTAlNGMRkwFwYDVQQKExBMaWJlcnR5IEFsbGlhbmNlMRQwEgYDVQQLEwtJT1AgVG
        VzdGVyczEiMCAGA1UEAxMZTGliZXJ0eSBUZXN0ZXJzIENlcnRpZmllcjEkMCIGCSqGSIb
        3DQEJARYVcnJvZHJpZ3VlekBuZW9zb2wubmV0MB4XDTAyMTIwNDE1NTg0NFoXDTEyMTIw
        MTE1NTg0NFowgasxCzAJBgNVBAYTAlVTMQswCQYDVQQHEwJTRjEkMCIGA1UEChMbTGliZ
        XJ0eSBBbGxpYW5jZSBlcmljc3Nvbi1hMSYwJAYDVQQLEx1JT1AgVGVzdGVycyBlcmljc3
        Nvbi1hIHNpZ25lcjEXMBUGA1UEAxMOZXJpY3Nzb24tYS5pb3AxKDAmBgkqhkiG9w0BCQE
```

EXAMPLE 9–1 SOAP Request for Authentication Assertion Using Web Browser Artifact Profile *(Continued)*

```
            WGXJyb2RyaWd1ZXpAZXJpY3Nzb24tYS5pb3AwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJ
            AoGBAPUoGYvJxQc5jzDnJ14TV6TaTbB3fH95ju24Z0y6HQxm6gXdJSAoWh7/AIes4UcV0
            9DC2kKS6Vow2YoXt2LIyH9HWH2tEUt1jS/PUeBHEWcW3tFezM6jh5GG5rCuVPZaW9eoGU
            bFPSzOPFKUAwdHUXSDWufY1KZ93IxhOBeZgg6VAgMBAAGjeTB3MEoGCWCGSAGG+EIBDQQ
            9FjtUaGlzIHNpZ25pbmcgY2VydCB3YXMgY3JlYXRlZCBmb3IgdGVzdGluZy4gRG8gbm90
            IHRydXN0 IGl0LjAJBgNVHRMEAjAAMBEGCWCGSAGG+EIBAQQEAwIEMDALBgNVHQ8EBAMC
            BsAwDQYJKoZIhvcNAQEEBQADgYEAR/HSgBpAprQwQVyWDE9pCaiduKv4/W/+hrdpXlVKS
            r6TIlg4ouDCQJNos7tNuG9ZAbfWtHvCss51N2cfAzfns/DKqxRqcsxzL5ZUBksPpmsDob
            oopUv6Xm8RFsi7yB9AGaVuqObeY/+m70nOu03O+FlMN3U1k2E3rOKXlU1noC0
            </ds:X509Certificate>
            </ds:X509Data>
        </ds:KeyInfo>
    </ds:Signature>
<samlp:AssertionArtifact>
    AAM1uXw6+f+jyA/4XuFHqPl7QDvc/LIQL9+t7YQtG1Gwk9bph0Adl+o+
</samlp:AssertionArtifact>
</samlp:Request>
</soap-env:Body>
</soap-env:Envelope>
```

In response to the request, SAML SOAP Receiver must return either a `<Response>` element within the body of another SOAP message or a SOAP fault code (error message) for every request received. The following sample is a response that contains an authentication assertion.

EXAMPLE 9–2 SOAP Response to SOAP Request for Web Browser Artifact Profile

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn
<soap-env:Envelope
    xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
    <soap-env:Header/>
    <soap-env:Body>
        <samlp:Response
            xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
            InResponseTo="RPCUk2ll+GVz+t1lLURp51oFvJXk"
            IssueInstant="2002-10-31T21:42:13Z"
            MajorVersion="1" MinorVersion="0"
            Recipient="http://localhost:8080/sp"
            ResponseID="LANWfL2xLybnc+BCwgY+p1/vIVAj">
            <samlp:Status>
                <samlp:StatusCode
                    xmlns:qns="urn:oasis:names:tc:SAML:1.0:protocol"
                    Value="qns:Success">
                </samlp:StatusCode>
```

EXAMPLE 9–2 SOAP Response to SOAP Request for Web Browser Artifact Profile *(Continued)*

```
</samlp:Status>
<saml:Assertion
    xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:lib="http://projectliberty.org/schemas/core/2002/12"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    AssertionID="SqMC8Hs2vJ7Z+t4UiLSmhKOSUO0U"
    InResponseTo="RPCUk2ll+GVz+t1lLURp51oFvJXk"
    IssueInstant="2002-10-31T21:42:13Z"
    Issuer="http://host:8080/idp"
    MajorVersion="1" MinorVersion="0"
    xsi:type="lib:AssertionType">
    <saml:Conditions
        NotBefore="2002-10-31T21:42:12Z"
        NotOnOrAfter="2002-10-31T21:42:43Z">
        <saml:AudienceRestrictionCondition>
            <saml:Audience>http://localhost:8080/sp</saml:Audience>
        </saml:AudienceRestrictionCondition>
    </saml:Conditions>
    <saml:AuthenticationStatement
        AuthenticationInstant="2002-10-31T21:42:13Z"
        AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
        xsi:type="lib:AuthenticationStatementType">
        <saml:Subject xsi:type="lib:SubjectType">
            <saml:NameIdentifier>
                C9FfGouQdBJ7bpkismYgd8ygeVb3PlWK
            </saml:NameIdentifier>
            <saml:SubjectConfirmation>
                <saml:ConfirmationMethod>
                    urn:oasis:names:tc:SAML:1.0:cm:artifact-01
                </saml:ConfirmationMethod>
            </saml:SubjectConfirmation>
            <lib:IDPProvidedNameIdentifier>
                C9FfGouQdBJ7bpkismYgd8ygeVb3PlWK
            </lib:IDPProvidedNameIdentifier>
        </saml:Subject>
    </saml:AuthenticationStatement>
    <ds:Signature>
        <ds:SignedInfo>
            <ds:CanonicalizationMethod
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
            </ds:CanonicalizationMethod>
        <ds:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
        </ds:SignatureMethod>
        <ds:Reference URI="">
```

EXAMPLE 9–2 SOAP Response to SOAP Request for Web Browser Artifact Profile *(Continued)*

```
                <ds:Transforms>
                <ds:Transform
                Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature">
                </ds:Transform>
                <ds:Transform
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                </ds:Transform>
                </ds:Transforms>
                <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
                </ds:DigestMethod>
                <ds:DigestValue>ZbscbqHTX9H8bBftRIWlG4Epv1A=</ds:DigestValue>
                </ds:Reference>
            </ds:SignedInfo>
            <ds:SignatureValue>
                H+q3nC3jUalj1uKUVkcC4iTFClxeZQIFF0nvHqPS5oZhtkBaDb9qI
                TA7gIkotaB584wXqTXwsfsuIrwT5uL3r85Rj7IF6NeCeiy3K0+z3u
                ewxyeZPz8wna449VNm0qNHYkgNak9ViNCp0/ks5MAttoPo2iLOfaK
                u3wWG6d1G+DM=
            </ds:SignatureValue>
        </ds:Signature>
    </saml:Assertion>
</samlp:Response>
</soap-env:Body>
</soap-env:Envelope>
```

Note – The entities requesting and responding with SAML must not include more than one SAML request or response per SOAP message. They must also not include any additional XML elements in the SOAP body.

Protecting SAML SOAP Receiver

The Access Manager administrator has the option of protecting the SAML SOAP Receiver. The available methods are:

- `NOAUTH`
- `BASICAUTH`
- `SSL`
- `SSLWITHBASICAUTH`

This value is configured as a subattribute of the Trusted Partners attribute in the SAML module. The default authentication type is `NOAUTH`. If `SSL` authentication is to be specified, it is configured in the `SOAPUrl` field with the `https` protocol. For more information, see "Trusted Partners" on page 183.

▼ To Configure Access Manager for Basic Authentication

Basic authentication allows a provider originating a request to authenticate itself by transmitting a username and password. The credentials are presented in response to a challenge from the provider to which the request is being sent. You need to configure Access Manager to support basic authentication using the following procedure.

1. **In the Access Manager Console, click the Federation tab.**
2. **Under Federation, click the SAML tab.**
3. **Select New under the Trusted Partners attribute.**
4. **Select the Web Browser Artifact Profile (Artifact) under Source and click Next.**
5. **Type a value for the Source ID attribute.**

   This is a 20–byte sequence (encoded using the Base64 format) that comes from the partner site. It is generally the same value as that used for the Site ID attribute when configuring "Site Identifiers" on page 182.
6. **Enter the SOAP Receiver URL for the site you are configuring as a value for the SOAP URL attribute.**

   General information on SOAP endpoints is in "SAML SOAP Receiver" on page 175.
7. **Select `BASICAUTH` or `SSLWITHBASICAUTH` (if the endpoint is configured with Secure Sockets Layer) as the authentication type.**
8. **Enter a user identifier for the user on the partner side being used to protect their SOAP Receiver.**
9. **Enter and reenter the password associated with the user on the partner side being used to protect their SOAP Receiver.**
10. **Click Finish to complete the configuration.**
11. **Click Save to save the configuration.**

SAML Attributes

The SAML module is configured by applying values to its attributes. `amSAML.xml` is the XML service file that defines the attributes. All SAML attributes are *global* in that the values applied to them are carried across the Access Manager configuration and inherited by every organization defined in the instance of Access Manager.

Note – For more information on service files, see *Sun Java System Access Manager 7 2005Q4 Administration Guide*.

Most attributes in the SAML module can be configured either through the Access Manager Console or the XML service file. "amSAML.xml Attributes" on page 181 lists the attributes that can *only* be configured by modifying the `amSAML.xml` file. "Console Attributes" on page 181 lists the attributes that can be configured using the console or the XML service file.

amSAML.xml Attributes

The following attributes can *only* be configured through the `amSAML.xml` file using the `amadmin` command-line interface.

- `iplanet-am-saml-cleanup-interval` is used to specify how often the internal thread is run to clean up expired assertions from the internal data store. The default is 180 seconds.
- `iplanet-am-saml-assertion-max-number` is used to specify the maximum number of assertions that the server can hold at one time. No new assertion is created if the maximum number is reached. The default value is 0, which means no limit.

▼ To Modify Attributes in the amSAML.xml File

1. **Duplicate the `amSAML.xml` service file and make any changes to the attributes.**
2. **Delete the old `amSAML.xml` service file.**
3. **Use `amadmin` to reload the newly modified `amSAML.xml` file.**

   For more information on `amadmin`, see the *Sun Java System Access Manager 7 2005Q4 Administration Guide*.

Console Attributes

The following SAML attributes can be configured by using the Access Manager Console or by modifying `amSAML.xml` as described in "amSAML.xml Attributes" on page 181. When viewed using the Console, the SAML attributes are separated into the following groups:

Properties Group

The attributes in the Properties group are as follows:

Target Specifier

This attribute assigns a name to the destination site URL value that is used in the redirects discussed in "Profile Types" on page 169. The default is `TARGET`. Only sites configured in the Trusted Partners attribute can be specified as a `TARGET`. For information, see "Trusted Partners" on page 183.

Site Identifiers

This attribute defines any site that is hosted by the server on which Access Manager is installed. A default value is defined for the host during installation (with values retrieved from `AMConfig.properties`), and a Site ID is automatically generated. Multiple entries are possible (for example, load balancing or multiple instances of Access Manager sharing the same Directory Server) although the default site identifier should always remain an entry.

Note – If configuring SAML for SSL (in both the source and destination site), ensure that the protocol defined in the Instance ID attribute is `HTTPS//`.

▼ To Configure a Site Identifier

You may also edit or duplicate entries already listed.

1 In the Access Manager Console, click the Federation tab.

2 Under Federation, click the SAML tab.

3 Select New under the Site Identifiers attribute.

4 Enter values for the following attributes:

Instance ID
: The value of this property is *protocol*`://`*host*`:`*port*.

Site ID
: This identifier is generated for each site, although the value will be the same for multiple servers behind a load balancer. To obtain this identifier manually, type the following at the command line:

```
% #java -classpath AM-classpath \ com.sun.identity.saml.common.SAMLSiteID
\protocol://host:port
```

For more information, see "com.sun.identity.saml.common Package" on page 189.

Issuer Name
The value of this property is *host*:*port*.

5 Click OK.

Trusted Partners

This attribute defines any trusted partner (remote to the server on which Access Manager is installed) that will be communicating with Access Manager.

Note – The trusted partner site must have a prearranged trust relationship with one or more of the sites configured in "Site Identifiers" on page 182.

Before configuring a trusted partner, you must determine the partner's role in the trust relationship. A trusted partner can be a *source site* (one that generates a single sign-on assertion) or a *destination site* (one that receives a single sign-on assertion). Following is the procedure for configuring a trusted partner.

▼ To Configure a Trusted Partner

The Trusted Partners attribute can contain one or more entries. Each entry is configured based on the site's defined role. For example, if the partner is the source site, this attribute is configured based on how it will send assertions. If the partner is the destination site, this attribute is configured based on which profile it uses to receive assertions.

1 In the Access Manager Console, click the Federation tab.

2 Under Federation, click the SAML tab.

3 Select New under the Trusted Partners attribute.

4 Select the role (Destination or Source) of the partner site that you are configuring by checking the appropriate profiles used to communicate with it and click Next.

Select Web Browser Artifact Profile or Web Browser Post Profile for either Destination, Source, or both, or SOAP Query for Source. The choices made dictate which of the attributes in the following steps need to be configured.

5 Type values for the Common Settings subattributes based on the selected roles.

Source ID
This is a 20–byte sequence (encoded using the Base64 format) that comes from the partner site. It is generally the same value as that used for the Site ID attribute when configuring "Site Identifiers" on page 182.

Target
: This is the domain of the partner site (with or without a port number). If you want to contact a web page that is hosted in this domain, the redirect URL is picked up from the values defined in "Trusted Partners" on page 183.

Note – If there are two defined entries for the same domain (one containing a port number and one without a port number), the entry with the port number takes precedence. For example, assume the following two trusted partner definitions: `target=sun.com` and `target=sun.com:8080`. If the principal is seeking `http://machine.sun.com:8080/index.html`, the second definition will be chosen.

Site Attribute Mapper
: The class is used to return a list of attribute values defined as `AttributeStatements` elements in an Authentication Assertion. A site attribute mapper needs to be implemented from one of the included interfaces:

  - com.sun.identity.saml.plugins.SiteAttributeMapper
  - com.sun.identity.saml.plugins.PartnerSiteAttributeMapper

  If no class is defined, no attributes will be included in the assertion. For more information, see "`SiteAttributeMapper` and `PartnerSiteAttributeMapper` Interfaces" on page 190.

Version
: The SAML version used (`1.0` or `1.1`) to send SAML requests. If this parameter is not defined, the following default values (defined in `AMConfig.properties`) are used:

  - `com.example.identity.saml.assertion.version=1.1`
  - `com.example.identity.saml.protocol.version=1.1`

Account Mapper
: The class that defines how the subject of an assertion is related to an identity at the destination site. The default is `com.sun.identity.saml.plugins.DefaultAccountMapper`. An account mapper needs to be implemented from one of the included interfaces:

  - `com.sun.identity.saml.plugins.AccountMapper`
  - `com.sun.identity.saml.plugins.PartnerAccountMapper`

  If no class is defined, no attributes will be included in the assertion. For more information, see "`AccountMapper` and `PartnerAccountMapper` Interfaces" on page 190.

Certificate
: A certificate alias that is used to verify the signature in an assertion when it is signed by the partner and the certificate cannot be found in the `KeyInfo` portion of the signed assertion.

Host List
: A list of the IP addresses, the DNS host name, or the `Certificate` name for all hosts within the partner site that can send requests to this authority. This list helps to ensure that the requestor is indeed the intended receiver of the artifact. If the requester is defined in this list, the interaction will continue. If the requester's information does not match any hosts defined in the host list, the request will be rejected.

Issuer
The creator of a generated assertion. The default syntax is *hostname*:*port*.

6 Type values for the Destination subattributes.

Artifact: SAML URL
The URL that points to the servlet that implements the Web Browser Artifact Profile. See "Web Browser Artifact Profile" on page 170.

Post: Post URL
The URL that points to the servlet that implements the Web Browser POST Profile. See "Web Browser POST Profile" on page 172.

SOAP Query: Attribute Mapper
The class that is used to obtain single sign-on information from a query. You need to implement an attribute mapper from the included interface. If no class is specified, the `DefaultAttributeMapper` will be used. For more information, see "`com.sun.identity.saml.plugins` Package" on page 189.

SOAP Query: Action Mapper
The class that is used to get single sign-on information and map partner actions to Access Manager authorization decisions. You need to implement an action mapper from the included interface. If no class is specified, the `DefaultActionMapper` will be used. For more information, see "`com.sun.identity.saml.plugins` Package" on page 189.

7 Type values for the Source subattributes.

Artifact: SOAP URL
The URL to the SAML SOAP Receiver. See "SAML SOAP Receiver" on page 175.

Authentication Type
Authentication types that can be used with SAML:

- `NOAUTH`
- `BASICAUTH`
- `SSL`
- `SSLWITHBASICAUTH`

This attribute is optional. If not specified, the default is `NOAUTH`. If `BASICAUTH` or `SSLWITHBASICAUTH` is specified, the Trusted Partners attribute is required and should be `HTTPS`. For more information, see "Trusted Partners" on page 183.

User
When Basic Authentication is chosen as the Authentication Type, the value of this attribute defines the user identifier of the partner being used to protect the partner's SOAP receiver.

User's Password
When Basic Authentication is chosen as the Authentication Type, the value of this attribute defines the password for the user identifier of the partner being used to protect the partner's SOAP receiver.

User's Password (reenter)
Reenter the password defined previously.

8 Click Finish to complete the configuration.

Target URLs

If the `TARGET` URL received through either profile is listed as a value of this attribute, the assertions received will be sent to the `TARGET` URL using an HTTP FORM POST.

Caution – Do not use test URLs or any other additional URLs in a POST.

To configure this attribute, type values for the following subattributes:

Protocol
Choose either `http` or `https`.

Server Name
The name of the server on which the TARGET URL resides, such as `www.sun.com`.

Port
The port number, such as `58080`.

Path
The URI, such as `/amserver/console`.

Assertion

The attributes in the Assertion group are as follows:

- "Assertion Timeout" on page 186
- "Assertion Skew Factor For notBefore Time" on page 186

Assertion Timeout

This attribute specifies the number of seconds before a timeout occurs on an assertion. The default is 420.

Assertion Skew Factor For notBefore Time

This attribute is used to calculate the `notBefore` time of an assertion. For example, if `IssueInstant` is `2002-09024T21:39:49Z`, and `Assertion Skew Factor For notBefore Time` is set to 300 seconds (180 is the default value), the `notBefore` attribute of the conditions element for the assertion would be `2002-09-24T21:34:49Z`.

Note – The total valid duration of an assertion is defined by the values set in both the `Assertion Timeout` and `Assertion Skew Factor For notBefore Time` attributes.

Artifact

The attributes in the Artifact group are as follows:

For more information about artifacts, see "Web Browser Artifact Profile" on page 170.

Artifact Timeout

This attribute specifies the period of time an assertion that is created for an artifact will be valid. The default is 400.

SAML Artifact Name

This attribute assigns a variable name to a SAML artifact. The artifact is bounded-size data that identifies an assertion and a source site. It is carried as part of a URL query string and conveyed by redirection to the destination site. The default name is `SAMLart`. Using the default `SAMLart`, the redirect query string could be `http://`*`host:port/deploy-URI`*`/SamlAwareServlet?TARGET=`*`target-URL`*`/&SAMLart=artifact123`.

Signing

The attributes in the Signing group are as follows:

Sign SAML Assertion

This attribute specifies whether all SAML assertions will be digitally signed (`XML DSIG`) before being delivered. Selecting the check box enables this feature.

Sign SAML Request

This attribute specifies whether all SAML requests will be digitally signed (`XML DSIG`) before being delivered. Selecting the check box enables this feature.

Sign SAML Response

This attribute specifies whether all SAML responses will be digitally signed (`XML DSIG`) before being delivered. Selecting the check box enables this feature.

Note – All SAML responses used by the Web Browser POST Profile are digitally signed whether or not this feature is enabled.

SAML API

Access Manager contains a SAML 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 Access Manager authorities to process external SAML requests and generate SAML responses. The packages include the following:

For more detailed information, 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.saml` Package

This package contains the `AssertionManager` and `SAMLClient` classes.

`AssertionManager` Class

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

- `createAssertion` creates an assertion with an authentication statement based on an Access Manager SSO Token ID.
- `createAssertionArtifact` creates an artifact that references an assertion based on an 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 9–3 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

Access Manager 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 interfaces:

- “`AccountMapper` and `PartnerAccountMapper` Interfaces” on page 190
- “`SiteAttributeMapper` and `PartnerSiteAttributeMapper` Interfaces” on page 190
- “`AttributeMapper` Interface” on page 190
- “`ActionMapper` Interface” on page 191

`AccountMapper` and `PartnerAccountMapper` Interfaces

`AccountMapper` and `PartnerAccountMapper` are interfaces that need 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 subattribute 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. Either SPI can be implemented although `PartnerAccountMapper` has one benefit over `AccountMapper`: it takes the whole assertion as a parameter, enabling the partner to define user account mapping based on attributes inside the assertion. The `AccountMapper` interface uses only the subject of the assertion as a parameter. 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 `AMConfig.properties` file logs additional information about the account mapper, for example, the user name and organization to which the mapper has been mapped. For more information about the `AMConfig.properties` file, see the *Sun Java System Access Manager 7 2005Q4 Developer's Guide*.

`SiteAttributeMapper` and `PartnerSiteAttributeMapper` Interfaces

`SiteAttributeMapper` and `PartnerSiteAttributeMapper` are interfaces that need 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 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 one of these mappers. This class would be used to obtain attributes, create the attribute statement, and insert the statement inside the assertion. Either SPI can be implemented although `PartnerSiteAttributeMapper` has one benefit over `SiteAttributeMapper`: there is an additional `targetURL` parameter that enables the partner to include different sets of attributes to target different applications.

Note – The default behavior is that no attribute statements are returned unless specified by the plug-in.

`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 Access Manager attributes. A default attribute mapper is provided if no other implementation is defined.

For more information, see "`AttributeQuery` Class" on page 191.

`ActionMapper` Interface

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

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

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

`AuthenticationQuery` Class

The `AuthenticationQuery` class represents a query for an authentication assertion. When an identity attempts to access a trusted partner web site, a SAML 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 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 Access Manager instance processing the query or a server that is trusted by the 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 Access Manager instance processing the query, or an assertion issued by a server that is trusted by the 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 9–4 AuthorizationDecisionQuery Code Sample

```
// testing getAssertion(authZQuery): no SC, with ni, with
// evidence(AssertionIDRef, authN, for this ni):
    String nameQualifier = "dc=iplanet,dc=com";
    String pName = "uid=amadmin,ou=people,dc=iplanet,dc=com";
    NameIdentifier ni = new NameIdentifier(pName, nameQualifier);
    Subject subject = new Subject(ni);
    String actionNamespace = "urn:test";
    // policy should be added to this resource with these
    // actions for the subject
    Action action1 = new Action(actionNamespace, "GET");
    Action action2 = new Action(actionNamespace, "POST");
    List actions = new ArrayList();
```

EXAMPLE 9–4 AuthorizationDecisionQuery Code Sample *(Continued)*

```
actions.add(action1);
actions.add(action2);
String resource = "http://www.sun.com:80";
eviSet = new HashSet();
// this assertion should contain authentication assertion for
// this subject and should be created by a trusted server
eviSet.add(eviAssertionIDRef3);
evidence = new Evidence(eviSet);
authzQuery = new AuthorizationDecisionQuery(eviSubject1, actions,
                    evidence, resource);
try {
    assertion = am.getAssertion(authzQuery, destID);
} catch (SAMLException e) {
    out.println("--failed. Exception:" + e);
}
```

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

All SAML 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 `AMConfig.properties` file. For information about these properties, see the *Sun Java System Access Manager 7 2005Q4 Developer's Guide*. For details on how to use the signature functionality, see "SAML Samples" on page 193.

SAML Samples

You can access several SAML-based samples from the Access Manager installation in */AccessManager-base*`/SUNWam/samples/saml`. These samples illustrate how the SAML service 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 */AccessManager-base*`/SUNWam/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 */AccessManager-base*`/SUNWam/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 */AccessManager-base*`/SUNWam/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 */AccessManager-base*`/SUNWam/samples/saml/xmlsig`.

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

◆ ◆ ◆ CHAPTER 10

Application Programming Interfaces

Sun Java System Access Manager provides a framework for identity federation and creating, discovering, and consuming identity web services. This framework includes a graphical user interface for Liberty-based web services as well as application programming interfaces (APIs). This chapter provides information on the APIs that do not have a corresponding graphical user interface (GUI).

This chapter covers the following topics:

Public Interfaces

The following list describes all of the public APIs you can use to deploy Liberty-enabled components or extend the core services. Packages that are part of a web service that has a GUI are described in the corresponding chapters of this book. Packages that are used solely on the back end are described in this chapter. Links to those sections are also provided. 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 10–1 Access Manager Public APIs

Package Name	Description
com.sun.identity.liberty.ws.authnsvc	Provides classes to manage the Authentication Web Service. See Chapter 5.

TABLE 10–1 Access Manager Public APIs *(Continued)*

Package Name	Description
`com.sun.identity.liberty.ws.authnsvc.mechanism`	Provides an interface to process incoming Simple Authentication and Security Layer (SASL) requests and generate SASL responses for the different SASL mechanisms. See Chapter 5.
`com.sun.identity.liberty.ws.authnsvc.protocol`	Provides classes to manage the Authentication Web Service protocol. See Chapter 5.
`com.sun.identity.liberty.ws.common`	Defines common classes used by many of the Access Manager Liberty-based web service components. See "Common Service Interfaces" on page 197.
`com.sun.identity.liberty.ws.common.wsse`	Provides an interface to parse and create an X.509 Certificate Token Profile. See "Common Service Interfaces" on page 197.
`com.sun.identity.liberty.ws.disco`	Provides interfaces to manage the Discovery Service. See Chapter 7.
`com.sun.identity.liberty.ws.disco.plugins`	Provides a plug-in interface for the Discovery Service. See Chapter 7.
`com.sun.identity.liberty.ws.dst`	Provides classes to implement an identity service on top of the Access Manager framework. See Chapter 6 for information about a service built using this API.
`com.sun.identity.liberty.ws.dst.service`	Provides a handler class that can be used by any generic identity data service. See Chapter 6 for information on data services.
`com.sun.identity.liberty.ws.interaction`	Provides classes to support the Liberty-based Interaction RequestRedirect Profile. See "Interaction Service" on page 201.
`com.sun.identity.liberty.ws.interfaces`	Provides interfaces common to all Access Manager Liberty-based web service components. See Chapter 6 and Chapter 7 for information about default implementations. See "Common Service Interfaces" on page 197 for more general information.
`com.sun.identity.liberty.ws.paos`	Provides classes for web applications to construct and process PAOS requests and responses. See "PAOS Binding" on page 203.
`com.sun.identity.liberty.ws.security`	Provides an interface to manage Liberty-based web service security mechanisms. See "Common Security API" on page 199.

TABLE 10–1 Access Manager Public APIs *(Continued)*

Package Name	Description
`com.sun.identity.liberty.ws.soapbinding`	Provides classes to construct SOAP requests and responses and to change the contact point for the SOAP binding. See Chapter 8.
`com.sun.identity.saml`	Provides an SPI in which proprietary XML signature implementations can be plugged in. See Chapter 9.
`com.sun.identity.saml.assertion`	Provides classes that manage assertions and profiles. See Chapter 9.
`com.sun.identity.saml.common`	Provides classes common to all SAML elements. See Chapter 9.
`com.sun.identity.saml.plugins`	Provides SPIs to integrate SAML into custom services. See Chapter 9.
`com.sun.identity.saml.protocol`	Provides classes that parse the XML messages used to exchange assertions and information. See Chapter 9.
`com.sun.identity.saml.xmlsig`	Provides an SPI in which proprietary XML signature implementations can be plugged in. See Chapter 9.
`com.sun.liberty`	Provides interfaces common to the Access Manager Federation Management module. See Chapter 3.

Common Service Interfaces

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

- "`com.sun.identity.liberty.ws.common` Package" on page 197
- "`com.sun.identity.liberty.ws.interfaces` Package" on page 198

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

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

TABLE 10–2 `com.sun.identity.liberty.ws.common` Classes

Class	Description
`LogUtil`	Defines methods that are used by the Liberty component of Access Manager to write logs.

TABLE 10–2 `com.sun.identity.liberty.ws.common` Classes *(Continued)*

Class	Description
`Status`	Represents a common status object.

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.

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

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

TABLE 10–3 `com.sun.identity.liberty.ws.interfaces` Interfaces

Interface	Description
`Authorizer`	Interface for identity service to check authorization of a WSC.
`ResourceIDMapper`	Interface used to map between a user ID and the Resource ID associated with it.
`ServiceInstanceUpdate`	Interface used to include a SOAP header (`ServiceInstanceUpdateHeader`) when sending a SOAP response.

`com.sun.identity.liberty.ws.interfaces.Authorizer` Interface

This interface, once implemented, can be used by each Liberty-based web service component for access control.

Note – The `com.sun.identity.liberty.ws.disco.plugins.DefaultDiscoAuthorizer` class is the implementation of this interface for the Discovery Service. For more information, see Chapter 7. The `com.sun.identity.liberty.ws.idpp.plugin.IDPPAuthorizer` class is the implementation for the Liberty Personal Profile Service. For more information, see Chapter 6.

The `Authorizer` interface enables a web service to check whether a web service consumer (WSC) is allowed to access the requested resource. When a WSC contacts a web service provider (WSP), the WSC conveys a sender identity and an invocation identity. Note that the *invocation identity* is always the subject of the SAML assertion. These conveyances enable the WSP to make an authorization decision based on one or both identities. The Access Manager Policy Service performs the authorization based on defined policies.

Note – See the *Sun Java System Access Manager 7 2005Q4 Technical Overview* for more information about policy management, single sign-on, and user sessions. See the *Sun Java System Access Manager 7 2005Q4 Administration Guide* for information about creating policy.

`com.sun.identity.liberty.ws.interfaces.ResourceIDMapper` Interface

This interface is used to map a user DN to the resource identifier associated with it. Access Manager provides implementations of this interface.

- `com.sun.identity.liberty.ws.disco.plugins.Default64ResourceIDMapper` assumes the Resource ID format to be: *providerID* + "/" + *the Base64 encoded userIDs*.
- `com.sun.identity.liberty.ws.disco.plugins.DefaultHexResourceIDMapper` assumes the Resource ID format to be: *providerID* + "/" + *the hex string of userID*.
- `com.sun.identity.liberty.ws.idpp.plugin.IDPPResourceIDMapper` assumes the Resource ID format to be: *providerID* + "/" + *the Base64 encoded userIDs*.

A different implementation of the interface may be developed. The implementation class should be given to the provider that hosts the Discovery Service. The mapping between the *providerID* and the implementation class can be configured through the Classes For ResourceIDMapper Plugin attribute.

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

The `com.sun.identity.liberty.ws.security` package includes the `SecurityTokenProvider` interface for managing Web Service Security (WSS) type tokens. The following table describes the classes used to manage Liberty-based security mechanisms.

TABLE 10–4 `com.sun.identity.liberty.ws.security` Classes

Class	Description
`ProxySubject`	Represents the identity of a proxy, the confirmation key, and confirmation obligation the proxy must possess and demonstrate for authentication purposes.

TABLE 10–4 `com.sun.identity.liberty.ws.security` Classes *(Continued)*

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

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

`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 10–5 `com.sun.identity.liberty.ws.common.wsse` Classes

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

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

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, Access Manager has implemented the Interaction RequestRedirect Profile. In this profile, the WSP requests the connecting WSC to redirect the user agent (principal) to an interaction resource (URL) at the WSP. When the user agent sends an HTTP request to get the URL, the WSP has the opportunity to present one or more pages to the principal with questions for other information. After the WSP obtains the information it needs to serve the WSC, it redirects the user agent back to the WSC, which can now reissue its original request to the WSP.

Configuring the Interaction Service

While there is no XML service file for the Interaction Service, this service does have properties. The properties are configured upon installation in the `AMConfig.properties` file located in */AccessManager-base*/SUNWam/lib and are described in the following table.

TABLE 10–6 Interaction Service Properties in `AMConfig.properties`

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

TABLE 10–6 Interaction Service Properties in `AMConfig.properties` *(Continued)*

Property	Description
`com.sun.identity.liberty.interaction.wscSpecifiedMaxInteractionTime`	Indicates the maximum length of time (in seconds) the WSC is willing to wait for the WSP to complete its portion of the interaction. The WSP will not initiate an interaction if the interaction is likely to take more time than . For example, the WSP receives a request where this property is set to a maximum 30 seconds. If the WSP property `com.sun.identity.liberty.interaction.wspRedirectTime` is set to 40 seconds, the WSP returns a SOAP fault (`timeNotSufficient`), indicating that the time is insufficient for interaction.
`com.sun.identity.liberty.interaction.wscWillEnforceHttpsCheck`	Indicates whether the WSC will enforce HTTPS in redirected URLs. The Liberty Alliance Project specifications state that, the value of this property is always `yes`, which indicates that the WSP will not redirect the user when the value of `redirectURL` (specified by the WSP) is not an HTTPS URL. The `false` value is primarily meant for ease of deployment in a phased manner.
`com.sun.identity.liberty.interaction.wspWillRedirect`	Initiates an interaction to get user consent for something or to collect additional data. This property indicates whether the WSP will redirect the user for consent. The default value is `yes`.
`com.sun.identity.liberty.interaction.wspWillRedirectForData`	Initiates an interaction to get user consent for something or to collect additional data. This property indicates whether the WSP will redirect the user to collect additional data. The default value is `yes`.
`com.sun.identity.liberty.interaction.wspRedirectTime`	Indicates the length of time (in seconds) that the WSP expects to take to complete an interaction and return control back to the WSC. For example, the WSP receives a request indicating that the WSC will wait a maximum 30 seconds (set in `com.sun.identity.liberty.interaction.wscSpecifiedMaxInteractionTime`) for interaction. If the `wspRedirectTime` is set to 40 seconds, the WSP returns a SOAP fault (`timeNotSufficient`), indicating that the time is insufficient for interaction.
`com.sun.identity.liberty.interaction.wspWillEnforceHttpsCheck`	Indicates whether the WSP will enforce a HTTPS `returnToURL` specified by the WSC. The Liberty Alliance Project specifications state that the value of this property is always `yes`. The `false` value is primarily meant for ease of deployment in a phased manner.

TABLE 10–6 Interaction Service Properties in `AMConfig.properties` *(Continued)*

Property	Description
`com.sun.identity.liberty.interaction.wspWillEnforceReturnToHostEqualsRequestHost`	Indicates whether the WSP would enforce the address values of `returnToHost` and `requestHost` if they are the same. The Liberty Alliance Project specifications state that the value of this property is always `yes`. The `false` value is primarily meant for ease of deployment in a phased manner.
`com.sun.identity.liberty.interaction.htmlStyleSheetLocation`	Points to the location of the style sheet that is used to render the interaction page in HTML.
`com.sun.identity.liberty.interaction.wmlStyleSheetLocation`	Points to the location of the style sheet that is used to render the interaction page in WML.

Interaction Service API

The 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 10–7 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 Java API Reference in */AccessManager-base*`/SUNWam/docs` or on docs.sun.com.

PAOS Binding

Access Manager has implemented the optional *Liberty Reverse HTTP Binding for SOAP Specification*. This specification defines a message exchange protocol that permits an HTTP client to be a SOAP responder. HTTP clients are no longer necessarily equipped with HTTP servers. For example, mobile terminals and personal computers contain web browsers yet they do not operate HTTP servers. These clients, though, can use their browsers to interact with an identity service, possibly a personal profile service or a calendar service. These identity services could also be

beneficial when the client devices interact with an HTTP server. The use of PAOS makes it possible to exchange information between user agent-hosted services and remote servers. This is why the reverse HTTP for SOAP binding is also known as PAOS; the spelling of SOAP is reversed.

Comparison of PAOS and SOAP

In a typical SOAP binding, an HTTP client interacts with an identity service through a client request and a server response. For example, a cell phone user (client) can contact the phone service provider (service) to retrieve stock quotes and weather information. The service verifies the user's identity and responds with the requested information.

In a reverse HTTP for SOAP binding, the phone service provider plays the client role, and the cell phone client plays the server role. The initial SOAP request from the server is actually bound to an HTTP response. The subsequent response from the client is bound to a request.

PAOS Binding API

The 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 Java API Reference in */AccessManager-base*`/SUNWam/docs` or on docs.sun.com.

TABLE 10–8 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.
`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.

TABLE 10–8 PAOS Binding Classes *(Continued)*

Class	Description
`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.

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.

PAOS Binding Sample

A sample that demonstrates PAOS service interaction between an HTTP client and server is provided in the */AccessManager-base*`/SUNWam/samples/phase2/paos` directory. The PAOS client is a servlet, and the PAOS server is a stand-alone Java program. 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. The following code example is the PAOS client servlet.

EXAMPLE 10–1 PAOS Client Servlet From PAOS Sample

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

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

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());
```

EXAMPLE 10–1 PAOS Client Servlet From PAOS Sample *(Continued)*

```
    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);
  }

  public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

      PAOSResponse paosRes = null;
      try {
    paosRes = new PAOSResponse(req);
      } catch (PAOSException pe) {
    pe.printStackTrace();
    throw new ServletException(pe.getMessage());
      }

      System.out.println("PAOS response from User Agent side ---------------->");
      System.out.println(paosRes.toString());

      System.out.println("Data output after parsing ---------------->");

      String dataStr = null;
      try {
```

EXAMPLE 10–1 PAOS Client Servlet From PAOS Sample *(Continued)*

```
    dataStr = paosRes.getPPResponseStr();
      } catch (PAOSException paose) {
    paose.printStackTrace();
    throw new ServletException(paose.getMessage());
      }
      System.out.println(dataStr);

      String msg = "Got the data: \\n" + dataStr;

      res.setContentType("text/plain");
      res.setContentLength(1+msg.length());

      PrintWriter out = new PrintWriter(res.getOutputStream());

      out.println(msg);

      out.close();
  }
}
```

See Appendix A for information about all the sample code and files included with Access Manager.

◆ ◆ ◆ APPENDIX A

Liberty-based and SAML Samples

Sun Java System Access Manager contains a number of samples that make use of the Access Manager implementation of the Liberty Alliance Project specifications. This appendix contains information about the samples. The samples are located in */AccessManager-base*/SUNWam/samples. This directory includes samples for the entire Access Manager product as well as two directories specific to the Liberty-based features: liberty and phase2.

This appendix covers the following samples:

Federation Framework Samples

Access Manager 2005Q4 supports the *Liberty Alliance Identity Federation Framework 1.2 Specifications*. The Federation Framework samples are located in */AccessManager-base*/SUNWam/samples/liberty. To demonstrate the different Liberty-based federation protocols featured in Access Manager, three sample applications are included. They are located in the following subdirectories:

sample1 Directory

The sample1 directory provides a collection of files to configure a basic environment for creating and managing a 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 Access Manager.

TABLE A–1 Configuration Information for `sample1` Servers

Variable Placeholder	Host Name	Components Deployed on This Host
machine1	`www.sp1.com`	■ Service Provider ■ Web Service Consumer
machine2	`www.idp1.com`	■ 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 Chapter 4.

`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 a Sun Java System Web Server that is protected by a Sun Java System Policy Agent. As in "`sample1` Directory" on page 209, the SP and IDP are deployed and configured on different Access Manager installations. 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 case, though, all hosted providers are deployed on a single installation of Access Manager. You need to host the same IP address (the one on which Access Manager is installed) in four different DNS domains. Thus, four virtual server instances are created on a 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 on the same machine as the 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 Chapter 4.

Web Services Framework Samples

Access Manager 2005Q4 supports both the *Liberty Alliance Identity Web Services Framework 1.0 Specifications* and the *Liberty Alliance Identity Services Interface Specifications 1.0*. The web services samples are located in */AccessManager-base*`/SUNWam/samples/phase2`. To demonstrate the different Liberty-based web services protocols featured in Access Manager, four sample applications are included. They are located in the following sub-directories:

`wsc` Directory

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 Access Manager installed, and running, and Liberty-enabled. Completing the procedure in "`sample1` Directory" on page 209 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 Chapter 6 and Chapter 7.

`sis-ep` Directory

The `sis-ep` directory contains a collection of files to develop, deploy, and invoke a new Liberty-based web service to Access Manager. The sample implements the Liberty Employee Profile Service.

Note – Before implementing this sample, you must have two instances of Access Manager installed, and running, and Liberty-enabled. Completing the procedure in "sample1 Directory" on page 209 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 Chapter 6

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. After running the sample, you will see the output from the PAOSServer program.

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 `Readme.html` file in the sample directory provides detailed steps on how to deploy and configure this sample. In addition, see "PAOS Binding Sample" on page 205.

Note – In an actual deployment, the server-side code would be developed by a service provider.

authnsvc Directory

The `authnsvc` directory contains a collection of files to illustrate the use of the Access Manager Authentication Web Service. This sample program authenticates against the service and extracts the resource offering of a discovery bootstrap. The `Readme.html` file in the sample directory provides detailed steps on how to deploy and configure this sample. In addition, see Chapter 5

SAML Samples

For information on the samples related to the SAML component of Access Manager, see "SAML Samples" on page 193.

◆ ◆ ◆ APPENDIX B

Service Schema Files

This appendix contains some of the XML Schema Definition (XSD) files that are discussed in this guide.

This appendix contains the following sections:

XSD Overview

The purpose of an eXtensible Markup Language (XML) schema is to describe the structure of an XML document. The XML schema language is referred to as XML Schema Definition (XSD).

Note – XSD is an XML-based alternative to the Document Type Definition (DTD). A DTD also describes the structure of an XML document, but it is not in the XML format.

The XSD files in this appendix specify the information that its corresponding service can host by defining the data and data structure. Typically, this structure is hierarchical and has one root node. Individual branches of the structure can be accessed separately, and the whole structure can be accessed by pointing to the root node. The data might be stored in implementation-specific ways, However, the data will be exposed by the service using the XML schema (specified here) and the Web Services Description Language (WSDL) definition of the service type (not specified in this documentation set). The XSD files in this appendix are reproduced here for your convenience. These files and a number of other XSD files are also available on the Liberty Alliance Project web site (`http://www.projectliberty.org/resources/specifications.php`).

SOAP Binding Schema

Following is a reproduction of `liberty-idwsf-soap-binding-v1.1.xsd`, the XSD file that accompanies the *Liberty ID-WSF SOAP Binding Specification* as discussed in Chapter 8.

EXAMPLE B–1 SOAP Binding XSD File

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:sb:2004-04"
    xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:sb-ext="urn:liberty:sb:2004-04"
    xmlns:lib="urn:liberty:iff:2003-08"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="urn:liberty:sb:2004-04"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">

    <!-- Author: John Kemp -->
    <!-- Last editor: $Author: dgreenspon $ -->
    <!-- $Date: 2004/08/02 19:25:27 $ -->
    <!-- $Revision: 1.1 $ -->

    <xs:import
        namespace="http://schemas.xmlsoap.org/soap/envelope/"
        schemaLocation="http://schemas.xmlsoap.org/soap/envelope/"/>

    <xs:import
        namespace="urn:liberty:iff:2003-08"
        schemaLocation="liberty-idff-protocols-schema-v1.2.xsd"/>

    <xs:include schemaLocation="liberty-idwsf-utility-1.0-errata-v1.0.xsd"/>

    <xs:annotation>
        <xs:documentation>
            Liberty ID-WSF SOAP Binding Specification Extension XSD
        </xs:documentation>
        <xs:documentation>
            The source code in this XSD file was excerpted verbatim from:

            Liberty ID-WSF SOAP Binding Specification
            Version 1.1
            April 2004

            Copyright (c) 2004 Liberty Alliance participants, see
            http://www.projectliberty.org/specs/idwsf_copyrights.html
        </xs:documentation>
    </xs:annotation>
```

EXAMPLE B–1 SOAP Binding XSD File *(Continued)*

```
    <xs:complexType name="CredentialsContextType">
        <xs:sequence>
            <xs:element ref="lib:RequestAuthnContext" minOccurs="0"/>
            <xs:element name="SecurityMechID" type="xs:anyURI" minOccurs="0"
                maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="optional"/>
        <xs:attribute ref="S:mustUnderstand" use="optional"/>
        <xs:attribute ref="S:actor" use="optional"/>
    </xs:complexType>

    <xs:element name="CredentialsContext" type="CredentialsContextType"/>

    <xs:complexType name="ServiceInstanceUpdateType">
        <xs:sequence>
            <xs:element name="SecurityMechID" type="xs:anyURI" minOccurs="0"
                maxOccurs="unbounded"/>
            <xs:element name="Credential" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                   <xs:sequence>
                       <xs:any namespace="##any" processContents="lax"/>
                   </xs:sequence>
                   <xs:attribute name="notOnOrAfter" type="xs:dateTime" use="optional"/>
                </xs:complexType>
            </xs:element>
            <xs:element name="Endpoint" type="xs:anyURI" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="optional"/>
        <xs:attribute ref="S:mustUnderstand" use="optional"/>
        <xs:attribute ref="S:actor" use="optional"/>
    </xs:complexType>

    <xs:element name="ServiceInstanceUpdate" type="ServiceInstanceUpdateType"/>

    <xs:complexType name="TimeoutType">
        <xs:attribute name="maxProcessingTime" type="xs:integer" use="required"/>
        <xs:attribute name="id" type="xs:ID" use="optional"/>
        <xs:attribute ref="S:mustUnderstand" use="optional"/>
        <xs:attribute ref="S:actor" use="optional"/>
    </xs:complexType>

    <xs:element name="Timeout" type="TimeoutType"/>

</xs:schema>
```

EXAMPLE B–1 SOAP Binding XSD File *(Continued)*

Personal Profile Schema

Following is a reproduction of `liberty-idsis-pp-v1.0.xsd`, the XSD file that accompanies the *Liberty ID-SIS Personal Profile Service Specification* as discussed in Chapter 6.

EXAMPLE B–2 Personal Profile Service XSD File

```
<!-- 2003-11-02-->
<xs:schema targetNamespace="urn:liberty:id-sis-pp:2003-08" xmlns="urn:liberty:id-sis-pp:2003-08"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
elementFormDefault="qualified" version="1.0">
  <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
      schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
  <xs:annotation>
    <xs:documentation>Title: Liberty ID-WSF-SIS Personal Profile Services Schema
    </xs:documentation>
    <xs:documentation>The source code in this XSD file was excerpted verbatim from:

Liberty Liberty ID-SIS Personal Profile Service Specification
Version 1.2
12th November 2003

Copyright (c) 2003 Liberty Alliance participants, see
https://www.projectliberty.org/specs/idwsf_copyrights.html
</xs:documentation>
  </xs:annotation>
  <xs:include schemaLocation="liberty-idwsf-dst-v1.0.xsd"/>
  <xs:include schemaLocation="liberty-idwsf-dst-dt-v1.0.xsd"/>
  <xs:complexType name="KeyInfoType" mixed="true">
    <xs:complexContent mixed="true">
      <xs:extension base="ds:KeyInfoType">
        <xs:attribute ref="modificationTime"/>
        <xs:attribute ref="ACC"/>
        <xs:attribute ref="ACCTime"/>
        <xs:attribute ref="modifier"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:simpleType name="SelectType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:element name="PP" type="PPType"/>
```

EXAMPLE B–2 Personal Profile Service XSD File *(Continued)*

```
<xs:complexType name="PPType">
  <xs:sequence>
    <xs:element ref="InformalName" minOccurs="0"/>
    <xs:element ref="LInformalName" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="CommonName" minOccurs="0"/>
    <xs:element ref="LegalIdentity" minOccurs="0"/>
    <xs:element ref="EmploymentIdentity" minOccurs="0"/>
    <xs:element ref="AddressCard" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="MsgContact" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Facade" minOccurs="0"/>
    <xs:element ref="Demographics" minOccurs="0"/>
    <xs:element ref="SignKey" minOccurs="0"/>
    <xs:element ref="EncryptKey" minOccurs="0"/>
    <xs:element ref="EmergencyContact" minOccurs="0"/>
    <xs:element ref="LEmergencyContact" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Extension" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="commonAttributes"/>
</xs:complexType>
<xs:element name="InformalName" type="DSTString"/>
<xs:element name="LInformalName" type="DSTLocalizedString"/>
<xs:element name="CommonName" type="CommonNameType"/>
<xs:complexType name="CommonNameType">
  <xs:sequence>
    <xs:element ref="CN" minOccurs="0"/>
    <xs:element ref="LCN" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="AltCN" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="LAltCN" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="AnalyzedName" minOccurs="0"/>
    <xs:element ref="Extension" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="commonAttributes"/>
</xs:complexType>
<xs:element name="CN" type="DSTString"/>
<xs:element name="LCN" type="DSTLocalizedString"/>
<xs:element name="AltCN" type="DSTString"/>
<xs:element name="LAltCN" type="DSTLocalizedString"/>
<xs:element name="AnalyzedName" type="AnalyzedNameType"/>
<xs:complexType name="AnalyzedNameType">
  <xs:sequence>
    <xs:element ref="PersonalTitle" minOccurs="0"/>
    <xs:element ref="LPersonalTitle" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="FN" minOccurs="0"/>
    <xs:element ref="LFN" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="SN" minOccurs="0"/>
    <xs:element ref="LSN" minOccurs="0" maxOccurs="unbounded"/>
```

EXAMPLE B–2 Personal Profile Service XSD File *(Continued)*

```
    <xs:element ref="MN" minOccurs="0"/>
    <xs:element ref="LMN" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Extension" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="nameScheme" type="xs:anyURI" use="optional"/>
  <xs:attributeGroup ref="commonAttributes"/>
</xs:complexType>
<xs:element name="PersonalTitle" type="DSTString"/>
<xs:element name="LPersonalTitle" type="DSTLocalizedString"/>
<xs:element name="FN" type="DSTString"/>
<xs:element name="LFN" type="DSTLocalizedString"/>
<xs:element name="SN" type="DSTString"/>
<xs:element name="LSN" type="DSTLocalizedString"/>
<xs:element name="MN" type="DSTString"/>
<xs:element name="LMN" type="DSTLocalizedString"/>
<xs:element name="LegalIdentity" type="LegalIdentityType"/>
<xs:complexType name="LegalIdentityType">
  <xs:sequence>
    <xs:element ref="LegalName" minOccurs="0"/>
    <xs:element ref="LLegalName" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="AnalyzedName" minOccurs="0"/>
    <xs:element ref="VAT" minOccurs="0"/>
    <xs:element ref="AltID" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="DOB" minOccurs="0"/>
    <xs:element ref="Gender" minOccurs="0"/>
    <xs:element ref="MaritalStatus" minOccurs="0"/>
    <xs:element ref="Extension" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="commonAttributes"/>
</xs:complexType>
<xs:element name="LegalName" type="DSTString"/>
<xs:element name="LLegalName" type="DSTLocalizedString"/>
<xs:element name="VAT" type="VATType"/>
<xs:complexType name="VATType">
  <xs:sequence>
    <xs:element ref="IDValue"/>
    <xs:element ref="IDType" minOccurs="0"/>
    <xs:element ref="Extension" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="commonAttributes"/>
</xs:complexType>
<xs:element name="IDValue" type="DSTString"/>
<xs:element name="IDType" type="DSTURI"/>
<xs:element name="AltID" type="AltIDType"/>
<xs:complexType name="AltIDType">
  <xs:sequence>
```

EXAMPLE B–2 Personal Profile Service XSD File *(Continued)*

```
    <xs:element ref="IDValue"/>
    <xs:element ref="IDType" minOccurs="0"/>
    <xs:element ref="Extension" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="commonAttributes"/>
</xs:complexType>
<xs:element name="DOB" type="DSTDate"/>
<xs:element name="Gender" type="DSTURI"/>
<xs:element name="MaritalStatus" type="DSTURI"/>
<xs:element name="EmploymentIdentity" type="EmploymentIdentityType"/>
<xs:complexType name="EmploymentIdentityType">
  <xs:sequence>
    <xs:element ref="JobTitle" minOccurs="0"/>
    <xs:element ref="LJobTitle" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="O" minOccurs="0"/>
    <xs:element ref="LO" minOccurs="0"/>
    <xs:element ref="AltO" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="AltLO" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Extension" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="commonAttributes"/>
</xs:complexType>
<xs:element name="JobTitle" type="DSTString"/>
<xs:element name="LJobTitle" type="DSTLocalizedString"/>
<xs:element name="O" type="DSTString"/>
<xs:element name="LO" type="DSTLocalizedString"/>
<xs:element name="AltO" type="DSTString"/>
<xs:element name="AltLO" type="DSTLocalizedString"/>
<xs:element name="AddressCard" type="AddressCardType"/>
<xs:complexType name="AddressCardType">
  <xs:sequence>
    <xs:element ref="AddrType" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Address" minOccurs="0"/>
    <xs:element ref="Nick" minOccurs="0"/>
    <xs:element ref="LNick" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="LComment" minOccurs="0"/>
    <xs:element ref="Extension" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="commonAttributes"/>
</xs:complexType>
<xs:element name="AddrType" type="DSTURI"/>
<xs:element name="Address" type="AddressType"/>
<xs:complexType name="AddressType">
  <xs:sequence>
    <xs:element ref="PostalAddress" minOccurs="0"/>
    <xs:element ref="LPostalAddress" minOccurs="0" maxOccurs="unbounded"/>
```

EXAMPLE B–2 Personal Profile Service XSD File *(Continued)*

```
    <xs:element ref="PostalCode" minOccurs="0"/>
    <xs:element ref="L" minOccurs="0"/>
    <xs:element ref="LL" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="St" minOccurs="0"/>
    <xs:element ref="LSt" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="C" minOccurs="0"/>
    <xs:element ref="Extension" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="commonAttributes"/>
</xs:complexType>
<xs:element name="PostalAddress" type="DSTString"/>
<xs:element name="LPostalAddress" type="DSTLocalizedString"/>
<xs:element name="PostalCode" type="DSTString"/>
<xs:element name="L" type="DSTString"/>
<xs:element name="LL" type="DSTLocalizedString"/>
<xs:element name="St" type="DSTString"/>
<xs:element name="LSt" type="DSTLocalizedString"/>
<xs:element name="C" type="DSTString"/>
<xs:element name="Nick" type="DSTString"/>
<xs:element name="LNick" type="DSTLocalizedString"/>
<xs:element name="LComment" type="DSTString"/>
<xs:element name="MsgContact" type="MsgContactType"/>
<xs:complexType name="MsgContactType">
  <xs:sequence>
    <xs:element ref="Nick" minOccurs="0"/>
    <xs:element ref="LNick" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="LComment" minOccurs="0"/>
    <xs:element ref="MsgType" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="MsgMethod" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="MsgTechnology" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="MsgProvider" minOccurs="0"/>
    <xs:element ref="MsgAccount" minOccurs="0"/>
    <xs:element ref="MsgSubaccount" minOccurs="0"/>
    <xs:element ref="Extension" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="commonAttributes"/>
</xs:complexType>
<xs:element name="MsgType" type="DSTURI"/>
<xs:element name="MsgMethod" type="DSTURI"/>
<xs:element name="MsgTechnology">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="DSTURI">
        <xs:attribute name="msgLimit" type="xs:integer" use="optional"/>
      </xs:extension>
    </xs:complexContent>
```

EXAMPLE B–2 Personal Profile Service XSD File *(Continued)*

```
    </xs:complexType>
  </xs:element>
  <xs:element name="MsgProvider" type="DSTString"/>
  <xs:element name="MsgAccount" type="DSTString"/>
  <xs:element name="MsgSubaccount" type="DSTString"/>
  <xs:element name="Facade" type="FacadeType"/>
  <xs:complexType name="FacadeType">
    <xs:sequence>
      <xs:element ref="MugShot" minOccurs="0"/>
      <xs:element ref="WebSite" minOccurs="0"/>
      <xs:element ref="NamePronounced" minOccurs="0"/>
      <xs:element ref="GreetSound" minOccurs="0"/>
      <xs:element ref="GreetMeSound" minOccurs="0"/>
      <xs:element ref="Extension" minOccurs="0"/>
    </xs:sequence>
    <xs:attributeGroup ref="commonAttributes"/>
  </xs:complexType>
  <xs:element name="MugShot" type="DSTURI"/>
  <xs:element name="WebSite" type="DSTURI"/>
  <xs:element name="NamePronounced" type="DSTURI"/>
  <xs:element name="GreetSound" type="DSTURI"/>
  <xs:element name="GreetMeSound" type="DSTURI"/>
  <xs:element name="Demographics" type="DemographicsType"/>
  <xs:complexType name="DemographicsType">
    <xs:sequence>
      <xs:element ref="DisplayLanguage" minOccurs="0"/>
      <xs:element ref="Language" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="Birthday" minOccurs="0"/>
      <xs:element ref="Age" minOccurs="0"/>
      <xs:element ref="TimeZone" minOccurs="0"/>
      <xs:element ref="Extension" minOccurs="0"/>
    </xs:sequence>
    <xs:attributeGroup ref="commonAttributes"/>
  </xs:complexType>
  <xs:element name="DisplayLanguage" type="DSTString"/>
  <xs:element name="Language" type="DSTString"/>
  <xs:element name="Birthday" type="DSTMonthDay"/>
  <xs:element name="Age" type="DSTInteger"/>
  <xs:element name="TimeZone" type="DSTString"/>
  <xs:element name="SignKey" type="KeyInfoType"/>
  <xs:element name="EncryptKey" type="KeyInfoType"/>
  <xs:element name="EmergencyContact" type="DSTString"/>
  <xs:element name="LEmergencyContact" type="DSTLocalizedString"/>
</xs:schema>
```

EXAMPLE B–2 Personal Profile Service XSD File *(Continued)*

Employee Profile Schema

Following is a reproduction of `liberty-idsis-ep-v1.0.xsd`, the XSD file that accompanies the *Liberty ID-SIS Employee Profile Service Specification* as discussed in Chapter 6.

EXAMPLE B–3 Employee Profile Service XSD Schema

```
<!-- Generated by gen-prof.pl  $Id: liberty-idsis-ep-v1.0.xsd,v 1.1 2004/08/02
19:25:27 dgreenspon Exp $from $Id: liberty-idsis-ep-v1.0.xsd,v 1.1 2004/08/02 19:25:27
dgreenspon Exp $ -->
<!-- adjust 2003-10-02 TDW: changed copyright -->
<xs:schema targetNamespace="urn:liberty:id-sis-ep:2003-08"
xmlns="urn:liberty:id-sis-ep:2003-08" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" version="1.0">
  <xs:annotation>
    <xs:documentation>Title: Liberty ID-SIS Employee Profile Services Schema</xs:documentation>
    <xs:documentation>The source code in this XSD file was excerpted verbatim from:

Liberty Liberty ID-SIS Employee Profile Service Specification
Version 1.2
12th November 2003

Copyright (c) 2003 Liberty Alliance participants, see
https://www.projectliberty.org/specs/idwsf_copyrights.html

</xs:documentation>
  </xs:annotation>
  <xs:include schemaLocation="liberty-idwsf-dst-v1.0.xsd"/>
  <xs:include schemaLocation="liberty-idwsf-dst-dt-v1.0.xsd"/>
  <xs:element name="EP" type="EPType"/>
  <xs:complexType name="EPType">
    <xs:sequence>
      <xs:element ref="EmployeeID" minOccurs="0"/>
      <xs:element ref="AltEmployeeID" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="DateOfHire" minOccurs="0"/>
      <xs:element ref="JobStartDate" minOccurs="0"/>
      <xs:element ref="EmployeeStatus" minOccurs="0"/>
      <xs:element ref="EmployeeType" minOccurs="0"/>
      <xs:element ref="InternalJobTitle" minOccurs="0"/>
      <xs:element ref="LInternalJobTitle" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="OU" minOccurs="0"/>
      <xs:element ref="LOU" minOccurs="0" maxOccurs="unbounded"/>
```

EXAMPLE B–3 Employee Profile Service XSD Schema *(Continued)*

```
    <xs:element ref="CorpCommonName" minOccurs="0"/>
    <xs:element ref="CorpLegalIdentity" minOccurs="0"/>
    <xs:element ref="ManagerEmployeeID" minOccurs="0"/>
    <xs:element ref="SubalternateEmployeeID" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Extension" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="commonAttributes"/>
</xs:complexType>
<xs:element name="EmployeeID" type="DSTString"/>
<xs:element name="AltEmployeeID" type="DSTString"/>
<xs:element name="DateOfHire" type="DSTDate"/>
<xs:element name="JobStartDate" type="DSTDate"/>
<xs:element name="EmployeeStatus" type="DSTURI"/>
<xs:element name="EmployeeType" type="DSTURI"/>
<xs:element name="InternalJobTitle" type="DSTString"/>
<xs:element name="LInternalJobTitle" type="DSTLocalizedString"/>
<xs:element name="OU" type="DSTString"/>
<xs:element name="LOU" type="DSTLocalizedString"/>
<xs:element name="CorpCommonName" type="CorpCommonNameType"/>
<xs:complexType name="CorpCommonNameType">
  <xs:sequence>
    <xs:element ref="CN" minOccurs="0"/>
    <xs:element ref="LCN" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="AltCN" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="LAltCN" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Extension" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="commonAttributes"/>
</xs:complexType>
<xs:element name="CN" type="DSTString"/>
<xs:element name="LCN" type="DSTLocalizedString"/>
<xs:element name="AltCN" type="DSTString"/>
<xs:element name="LAltCN" type="DSTLocalizedString"/>
<xs:element name="CorpLegalIdentity" type="CorpLegalIdentityType"/>
<xs:complexType name="CorpLegalIdentityType">
  <xs:sequence>
    <xs:element ref="LegalName" minOccurs="0"/>
    <xs:element ref="LLegalName" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="VAT" minOccurs="0"/>
    <xs:element ref="AltID" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Extension" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="commonAttributes"/>
</xs:complexType>
<xs:element name="LegalName" type="DSTString"/>
<xs:element name="LLegalName" type="DSTLocalizedString"/>
```

EXAMPLE B–3 Employee Profile Service XSD Schema *(Continued)*

```
  <xs:element name="VAT" type="VATType"/>
  <xs:complexType name="VATType">
    <xs:sequence>
      <xs:element ref="IDValue"/>
      <xs:element ref="IDType" minOccurs="0"/>
      <xs:element ref="Extension" minOccurs="0"/>
    </xs:sequence>
    <xs:attributeGroup ref="commonAttributes"/>
  </xs:complexType>
  <xs:element name="IDValue" type="DSTString"/>
  <xs:element name="IDType" type="DSTURI"/>
  <xs:element name="AltID" type="AltIDType"/>
  <xs:complexType name="AltIDType">
    <xs:sequence>
      <xs:element ref="IDValue"/>
      <xs:element ref="IDType" minOccurs="0"/>
      <xs:element ref="Extension" minOccurs="0"/>
    </xs:sequence>
    <xs:attributeGroup ref="commonAttributes"/>
  </xs:complexType>
  <xs:element name="ManagerEmployeeID" type="DSTString"/>
  <xs:element name="SubalternateEmployeeID" type="DSTString"/>
  <xs:simpleType name="SelectType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
</xs:schema>
```

Authentication Web Service Schema

Following is a reproduction of `liberty-idwsf-authn-svc-v1.0.xsd`, the XSD file that accompanies the *Liberty ID-WSF Authentication Service Specification* as discussed in Chapter 5.

EXAMPLE B–4 Authentication Web Service XSD File

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

<xs:schema
targetNamespace="urn:liberty:sa:2004-04"
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sa="urn:liberty:sa:2004-04"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

EXAMPLE B–4 Authentication Web Service XSD File *(Continued)*

```
xmlns:lib="urn:liberty:iff:2003-08"
xmlns:disco="urn:liberty:disco:2003-08"
xmlns="urn:liberty:sa:2004-04"
elementFormDefault="qualified"
attributeFormDefault="unqualified"
version="06">

<!-- Filename: lib-arch-authn-svc.xsd  -->
<!-- $Id: liberty-idwsf-authn-svc-v1.0.xsd,v 1.1 2004/08/02 19:25:27 dgreenspon Exp $ -->
<!-- Author: Jeff Hodges -->
<!-- Last editor: $Author: dgreenspon $ -->
<!-- $Date: 2004/08/02 19:25:27 $ -->
<!-- $Revision: 1.1 $ -->

  <xs:import
      namespace="urn:liberty:iff:2003-08"
      schemaLocation="liberty-idff-protocols-schema-v1.2.xsd"/>

  <xs:import
      namespace="urn:liberty:disco:2003-08"
      schemaLocation="liberty-idwsf-disco-svc-1.0-errata-v1.0.xsd"/>

  <xs:include schemaLocation="liberty-idwsf-utility-1.0-errata-v1.0.xsd"/>

  <xs:annotation>
      <xs:documentation>
          Liberty ID-WSF Authentication Service XSD
      </xs:documentation>
      <xs:documentation>
          The source code in this XSD file was excerpted verbatim from:
          Liberty ID-WSF Authentication Service Specification
          Version 1.0
          16 Feb 2004
          Copyright (c) 2003, 2004 Liberty Alliance participants,
          see http://www.projectliberty.org/specs/idwsf_copyrights.html
      </xs:documentation>
  </xs:annotation>

  <!-- SASLRequest and SASLResponse ID-* messages  -->

  <xs:element name="SASLRequest">
      <xs:complexType>
          <xs:sequence>

              <xs:element name="Data" minOccurs="0">
                  <xs:complexType>
```

EXAMPLE B–4 Authentication Web Service XSD File *(Continued)*

```
                    <xs:simpleContent>
                        <xs:extension base="xs:base64Binary"/>
                    </xs:simpleContent>
                </xs:complexType>
            </xs:element>

            <xs:element ref="lib:RequestAuthnContext"
                        minOccurs="0"/>

        </xs:sequence>

        <xs:attribute name="mechanism"
                      type="xs:string"
                      use="required"/>

        <xs:attribute name="authzID"
                      type="xs:string"
                      use="optional"/>

        <xs:attribute name="advisoryAuthnID"
                      type="xs:string"
                      use="optional"/>

        <xs:attribute name="id"
                      type="xs:ID"
                      use="optional"/>

    </xs:complexType>
</xs:element>


<xs:element name="SASLResponse">
    <xs:complexType>
        <xs:sequence>

            <xs:element ref="Status"/>

            <xs:element ref="PasswordTransforms" minOccurs="0"/>

            <xs:element name="Data" minOccurs="0">
                <xs:complexType>
                    <xs:simpleContent>
                        <xs:extension base="xs:base64Binary"/>
                    </xs:simpleContent>
                </xs:complexType>
            </xs:element>
```

EXAMPLE B–4 Authentication Web Service XSD File *(Continued)*

```
            <xs:element ref="disco:ResourceOffering"
                        minOccurs="0"
                        maxOccurs="unbounded"/>

            <xs:element name="Credentials" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:any namespace="##any"
                                processContents="lax"
                                minOccurs="0"
                                maxOccurs="unbounded"/>
                    </xs:sequence>
                 </xs:complexType>
             </xs:element>

         </xs:sequence>

         <xs:attribute name="serverMechanism"
                       type="xs:string"
                       use="optional"/>

         <xs:attribute name="id"
                       type="xs:ID"
                       use="optional"/>

    </xs:complexType>
</xs:element>

<!-- Password Transformations  -->

<xs:element name="PasswordTransforms">

    <xs:annotation>
        <xs:documentation>
            Contains ordered list of sequential password transformations
        </xs:documentation>
    </xs:annotation>

    <xs:complexType>
        <xs:sequence>

            <xs:element name="Transform" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
```

EXAMPLE B–4 Authentication Web Service XSD File *(Continued)*

```
            <xs:element name="Parameter"
                        minOccurs="0"
                        maxOccurs="unbounded">
                <xs:complexType>
                    <xs:simpleContent>
                        <xs:extension base="xs:string">
                            <xs:attribute name="name"
                                          type="xs:string"
                                          use="required"/>
                        </xs:extension>
                    </xs:simpleContent>
                </xs:complexType>
            </xs:element>

        </xs:sequence>

        <xs:attribute name="name"
                      type="xs:anyURI"
                      use="required"/>

        <xs:attribute name="id"
                      type="xs:ID"
                      use="optional"/>

    </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
</xs:element>

</xs:schema>
```

PAOS Binding Schema

Following is a reproduction of `liberty-paos-1.0-errata-v1.0.xsd`, the XSD file that accompanies the *Liberty Reverse HTTP Binding for SOAP Specification*. This XSD file describes the structure of PAOS requests and responses. PAOS binding is discussed in Chapter 10.

EXAMPLE B–5 Reverse HTTP Binding for SOAP XSD File

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:paos:2003-08" xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

EXAMPLE B–5 Reverse HTTP Binding for SOAP XSD File *(Continued)*

```
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns="urn:liberty:paos:2003-08"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:documentation>The source code in this XSD file was excerpted verbatim from:

Liberty Reverse HTTP Binding
Version 1.0
12th November 2003

Copyright (c) 2003 Liberty Alliance participants, see
https://www.projectliberty.org/specs/idwsf_copyrights.html

       </xs:documentation>
  </xs:annotation>
  <xs:import namespace="http://schemas.xmlsoap.org/soap/envelope/"
      schemaLocation="http://schemas.xmlsoap.org/soap/envelope/"/>
  <xs:include schemaLocation="liberty-utility-v1.0.xsd"/>
  <xs:element name="Request" type="RequestType"/>
  <xs:complexType name="RequestType">
    <xs:attribute name="responseConsumerURL" type="xs:anyURI" use="required"/>
    <xs:attribute name="service" type="xs:anyURI" use="required"/>
    <xs:attribute name="messageID" type="IDType" use="optional"/>
    <xs:attribute ref="S:mustUnderstand" use="required"/>
    <xs:attribute ref="S:actor" use="required"/>
  </xs:complexType>
  <xs:element name="Response" type="ResponseType"/>
  <xs:complexType name="ResponseType">
    <xs:attribute name="refToMessageID" type="IDType" use="optional"/>
    <xs:attribute ref="S:mustUnderstand" use="required"/>
    <xs:attribute ref="S:actor" use="required"/>
  </xs:complexType>
</xs:schema>
```

Metadata Description Schema

Following is a reproduction of `liberty-metadata-1.0-errata-v1.0.xsd`, the XSD file that accompanies the *Liberty Metadata Description and Discovery Specification*. This XSD file describes metadata, protocols for obtaining metadata, and resolution methods for discovering the location of metadata.

EXAMPLE B–6 Metadata Description and Discovery XSD File

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:liberty:metadata:2003-08" xmlns="urn:liberty:metadata:2003-08"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="1.0">
  <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
   schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
  <xs:import namespace="urn:oasis:names:tc:SAML:1.0:assertion"
   schemaLocation="oasis-sstc-saml-schema-assertion-1.1.xsd"/>
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
   schemaLocation="http://www.w3.org/2001/xml.xsd"/>
  <xs:include schemaLocation="liberty-utility-v1.0.xsd"/>
  <xs:annotation>
    <xs:documentation>XML Schema fom Metadata description and discovery protocols</xs:documentation>
    <xs:documentation>The source code in this XSD file was excerpted verbatim from:

Liberty Metadata Description and Discovery Specification
Version 1.0
12th November 2003

Copyright (c) 2003 Liberty Alliance participants, see
https://www.projectliberty.org/specs/idff_copyrights.html

</xs:documentation>
  </xs:annotation>
  <xs:simpleType name="entityIDType">
    <xs:restriction base="xs:anyURI">
      <xs:maxLength value="1024" id="maxlengthid"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:attribute name="libertyPrincipalIdentifier" type="entityIDType"/>
  <xs:attribute name="providerID" type="entityIDType"/>
  <xs:attribute name="validUntil" type="xs:dateTime"/>
  <xs:attribute name="cacheDuration" type="xs:duration"/>
  <xs:complexType name="additionalMetadataLocationType">
    <xs:simpleContent>
      <xs:extension base="xs:anyURI">
        <xs:attribute name="namespace" type="xs:anyURI"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="organizationType">
    <xs:sequence>
      <xs:element name="OrganizationName" type="organizationNameType" maxOccurs="unbounded"/>
      <xs:element name="OrganizationDisplayName" type="organizationDisplayNameType" maxOccurs="unbounded"/>
      <xs:element name="OrganizationURL" type="localizedURIType" maxOccurs="unbounded"/>
```

EXAMPLE B–6 Metadata Description and Discovery XSD File *(Continued)*

```
    <xs:element ref="Extension" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="organizationNameType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute ref="xml:lang"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="organizationDisplayNameType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute ref="xml:lang" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="localizedURIType">
  <xs:simpleContent>
    <xs:extension base="xs:anyURI">
      <xs:attribute ref="xml:lang" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="contactType">
  <xs:sequence>
    <xs:element name="Company" type="xs:string" minOccurs="0"/>
    <xs:element name="GivenName" type="xs:string" minOccurs="0"/>
    <xs:element name="SurName" type="xs:string" minOccurs="0"/>
    <xs:element name="EmailAddress" type="xs:anyURI" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="TelephoneNumber" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Extension" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute ref="libertyPrincipalIdentifier" use="optional"/>
  <xs:attribute name="contactType" type="attr.contactType" use="required"/>
</xs:complexType>
<xs:simpleType name="attr.contactType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="technical"/>
    <xs:enumeration value="administrative"/>
    <xs:enumeration value="billing"/>
    <xs:enumeration value="other"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="keyTypes">
  <xs:restriction base="xs:string">
```

EXAMPLE B–6 Metadata Description and Discovery XSD File *(Continued)*

```
    <xs:enumeration value="encryption"/>
    <xs:enumeration value="signing"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="providerDescriptorType">
  <xs:sequence>
    <xs:element name="KeyDescriptor" type="keyDescriptorType" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="SoapEndpoint" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="SingleLogoutServiceURL" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="SingleLogoutServiceReturnURL" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="FederationTerminationServiceURL" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="FederationTerminationServiceReturnURL" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="FederationTerminationNotificationProtocolProfile" type="xs:anyURI"
     minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="SingleLogoutProtocolProfile" type="xs:anyURI" minOccurs="0"
     maxOccurs="unbounded"/>
    <xs:element name="RegisterNameIdentifierProtocolProfile" type="xs:anyURI" minOccurs="0"
     maxOccurs="unbounded"/>
    <xs:element name="RegisterNameIdentifierServiceURL" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="RegisterNameIdentifierServiceReturnURL" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="NameIdentifierMappingProtocolProfile" type="saml:AuthorityBindingType"
     minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="NameIdentifierMappingEncryptionProfile" type="xs:anyURI" minOccurs="0"
     maxOccurs="unbounded"/>
    <xs:element name="Organization" type="organizationType" minOccurs="0"/>
    <xs:element name="ContactPerson" type="contactType" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="AdditionalMetaLocation" type="additionalMetadataLocationType"
     minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Extension" minOccurs="0"/>
    <xs:element ref="ds:Signature" minOccurs="0"/>
  </xs:sequence>
  <!--xs:attribute ref="providerID" use="required"/-->
  <xs:attribute name="protocolSupportEnumeration" type="xs:NMTOKENS" use="required"/>
  <xs:attribute name="id" type="xs:ID" use="optional"/>
  <xs:attribute ref="validUntil" use="optional"/>
  <xs:attribute ref="cacheDuration" use="optional"/>
</xs:complexType>
<!--added-->
<xs:element name="KeyDescriptor" type="keyDescriptorType"/>
<xs:complexType name="keyDescriptorType">
  <xs:sequence>
    <xs:element name="EncryptionMethod" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="KeySize" type="xs:integer" minOccurs="0"/>
    <xs:element ref="ds:KeyInfo" minOccurs="0"/>
    <xs:element ref="Extension" minOccurs="0"/>
  </xs:sequence>
```

EXAMPLE B–6 Metadata Description and Discovery XSD File *(Continued)*

```
  <xs:attribute name="use" type="keyTypes" use="optional"/>
</xs:complexType>
<!-- -->
<xs:element name="EntityDescriptor" type="entityDescriptorType"/>
<xs:group name="providerGroup">
  <xs:sequence>
    <xs:element name="IDPDescriptor" type="IDPDescriptorType" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="SPDescriptor" type="SPDescriptorType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:group>
<xs:complexType name="entityDescriptorType">
  <xs:sequence>
    <xs:choice>
      <xs:group ref="providerGroup"/>
      <xs:element name="AffiliationDescriptor" type="affiliationDescriptorType"/>
    </xs:choice>
    <xs:element name="ContactPerson" type="contactType" minOccurs="0"/>
    <xs:element name="Organization" type="organizationType" minOccurs="0"/>
    <xs:element ref="Extension" minOccurs="0"/>
    <xs:element ref="ds:Signature" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute ref="providerID" use="required"/>
  <xs:attribute name="id" type="xs:ID" use="optional"/>
  <xs:attribute ref="validUntil" use="optional"/>
  <xs:attribute ref="cacheDuration" use="optional"/>
</xs:complexType>
<xs:complexType name="SPDescriptorType">
  <xs:complexContent>
    <xs:extension base="providerDescriptorType">
      <xs:sequence>
        <xs:element name="AssertionConsumerServiceURL" maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:anyURI">
                <xs:attribute name="id" type="xs:ID" use="required"/>
                <xs:attribute name="isDefault" type="xs:boolean" default="false"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="AuthnRequestsSigned" type="xs:boolean"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="IDPDescriptorType">
```

EXAMPLE B–6 Metadata Description and Discovery XSD File *(Continued)*

```
    <xs:complexContent>
      <xs:extension base="providerDescriptorType">
        <xs:sequence>
          <xs:element name="SingleSignOnServiceURL" type="xs:anyURI"/>
          <xs:element name="SingleSignOnProtocolProfile" type="xs:anyURI" maxOccurs="unbounded"/>
          <xs:element name="AuthnServiceURL" type="xs:anyURI" minOccurs="0"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="EntitiesDescriptor" type="entitiesDescriptorType"/>
  <xs:complexType name="entitiesDescriptorType">
    <xs:sequence>
      <xs:element ref="EntityDescriptor" minOccurs="2" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="affiliationDescriptorType">
    <xs:sequence>
      <xs:element name="AffiliateMember" type="entityIDType" maxOccurs="unbounded"/>
      <xs:element ref="Extension" minOccurs="0"/>
      <xs:element name="KeyDescriptor" type="keyDescriptorType" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="ds:Signature" minOccurs="0"/>
    </xs:sequence>
    <!-- <xs:attribute name="affiliationID" type="entityIDType" use="required"/> -->
    <xs:attribute name="affiliationOwnerID" type="entityIDType" use="required"/>
    <xs:attribute ref="validUntil" use="optional"/>
    <xs:attribute ref="cacheDuration" use="optional"/>
    <xs:attribute name="id" type="xs:ID" use="optional"/>
  </xs:complexType>
</xs:schema>
```

Index

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

W

X