Sun Java™ System

Access Manager 6 Federation Management Guide

2005Q1

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

Part No: 817-7648

Contents

List of Figures

List of Tables

List of Code Examples

Preface

The *Sun Java™ System Access Manager 6 2005Q1 Federation Management Guide* provides information about the Federated Management module and related Web services in Sun Java™ System Access Manager 6 2005Q1 (formerly Sun™ ONE Identity Server). It includes an introduction to the Liberty Alliance Project's specifications and Access Manager's compliance with them. Instructions for enabling a Liberty-based environment, and summaries of the application programming interface (API) for extending the framework are also provided. This preface includes the following sections:

- Who Should Use This Guide
- Before You Read This Guide
- Conventions Used in This Guide
- Access Manager Documentation Set
- Related JES Product Documentation
- Accessing Sun Resources Online
- Contacting Sun Technical Support
- Related Third-Party Web Site References
- Sun Welcomes Your Feedback

Who Should Use This Guide

This *Federation Management Guide* is intended for use by IT professionals, network administrators and software developers who implement a Liberty-enabled identity management and web 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)
- SOAP (SOAP is no longer an acronym for the messaging protocol.)

Before You Read This Guide

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. You should be familiar with the documentation provided with Sun Java Enterprise System, which you can access online at:

`http://docs.sun.com/prod/entsys.05q1`

Because Sun Java System Directory Server is used as the data store in an Access Manager deployment, administrators should also be familiar with the documentation provided with that product. The latest Directory Server documentation can be accessed online at `http://docs.sun.com/coll/DirectoryServer_05q1`.

Conventions Used in This Guide

In the Access Manager documentation set, certain typographic conventions and terminology are used. These conventions are described in the following sections.

Typographic Conventions

The following table describes the typographic conventions used in this guide.

Table 1 Typographic Conventions

Typeface	Meaning	Examples
AaBbCc123 (Monospace)	API and language elements, HTML tags, web site URLs, command names, file names, directory path names, onscreen computer output, sample code.	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`
AaBbCc123 (Monospace bold)	What you type, when contrasted with onscreen computer output.	`% su` `Password:`
AaBbCc123 (Italic)	Book titles, new terms, words to be emphasized. A placeholder in a command or path name to be replaced with a real name or value.	Read Chapter 6 in the *User's Guide*. These are called *class* options. Do *not* save the file. The file is located in the *install-dir*`/bin` directory.

Symbols

The following table describes the symbol conventions used in this guide.

Table 2 Symbol Conventions

Symbol	Description	Example	Meaning
`[ ]`	Contains optional command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.

Table 2 Symbol Conventions *(Continued)*

Symbol	Description	Example	Meaning
>	Indicates menu item selection in a graphical user interface.	File > New > Templates	From the File menu, choose New. From the New submenu, choose Templates.

Default Paths and File Names

The following table describes the default paths and file names used in this guide:

Table 3 Default Paths and File Names

Term	Description
AccessManager_base	Represents the base installation directory for Access Manager. The Access Manager 2005Q1 default base installation and product directory depends on your specific platform: Solaris™ systems: `/opt/SUNWam` Linux systems: `/opt/sun/identity`
DirectoryServer_base	Represents the base installation directory for Sun Java System Directory Server. Refer to the product documentation for the specific path name.
ApplicationServer_base	Represents the base installation directory for Sun Java System Application Server. Refer to the product documentation for the specific path name.
WebServer_base	Represents the base installation directory for Sun Java System Web Server. Refer to the product documentation for the specific path name.

Shell Prompts

The following table describes the shell prompts used in this guide.

Table 4 Shell Prompts

Shell	Prompt
C shell on UNIX or Linux	*machine-name*%
C shell superuser on UNIX or Linux	*machine-name*#

Table 4 Shell Prompts

Shell	Prompt
Bourne shell and Korn shell on UNIX or Linux	$
Bourne shell and Korn shell superuser on UNIX or Linux	#
Windows command line	C:\

Access Manager Documentation Set

The Access Manager documentation consists of two sets:

- Access Manager Core Documentation
- Access Manager Policy Agent Documentation

NOTE For instructions on installing Access Manager, see the *Sun Java Enterprise System 2005Q1 Installation Guide* (http://docs.sun.com/doc/819-0056)

Access Manager Core Documentation

The Access Manager documentation set contains the following titles:

- The *Release Notes* (http://docs.sun.com/doc/817-7642) will be available online after the product is released. They gather 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.
- *Technical Overview* (http://docs.sun.com/doc/817-7643) provides an overview of how Access Manager components work together to consolidate identity management and to protect enterprise assets and web-based applications. It also explains basic Access Manager concepts and terminology.
- *Deployment Planning Guide* (http://docs.sun.com/doc/817-7644) provides information for planning an Access Manager deployment within an existing information technology infrastructure.
- *Migration Guide* (http://docs.sun.com/doc/817-7645) provides details on how to migrate existing data and Sun Java System product deployments to the latest version of Access Manager.

- *Performance Tuning Guide* (http://docs.sun.com/doc/817-7646) provides information on how to tune Access Manager and its related components for optimal performance.
- *Administration Guide* (http://docs.sun.com/doc/817-7647) describes how to use the Access Manager console as well as manage user and service data via the command line interface.
- *Federation Management Guide* (this guide) provides information about the Federation Management module and related Web services developed for Access Manager. These features are based on the Liberty Alliance Project (LAP) specifications available online at the LAP Web site, http://www.projectliberty.org/resources/specifications.php#box1.
- *Developer's Guide* (http://docs.sun.com/doc/817-7649) 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.
- *Developer's Reference* (http://docs.sun.com/doc/817-7650) provides summaries of data types, structures, and functions that make up the public Access Manager C APIs.
- *Java Specifications* (http://docs.sun.com/doc/817-7651) provides information on the implementation of Java packages in 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 2005Q1 documentation web site (http://docs.sun.com/prod/entsys.05q1). Updated documents will be marked with a revision date.

Access Manager Policy Agent Documentation

Documentation for the Access Manager policy agents is available at http://docs.sun.com/coll/S1_IdServPolicyAgent_21. Policy agents are developed on a different schedule than the server product itself. Therefore, the documentation set for the policy agents is available outside the core set of Access Manager documentation. The Policy Agent documentation set contains the following titles:

- *Web Policy Agents Guide* documents how to install and configure an Access Manager policy agent on various web and proxy servers. It also includes troubleshooting and information specific to each agent.

- *J2EE Policy Agents Guide* documents how to install and configure an Access Manager policy agent to protect a variety of hosted J2EE applications. It also includes troubleshooting and information specific to each agent.
- The *Release Notes* will be available online after the set of agents is released. There is generally one *Release Notes* file for each agent type release. The *Release Notes* gather 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.

Related JES Product Documentation

Useful information can be found at the following locations:

- Directory Server documentation:
  `http://docs.sun.com/coll/DirectoryServer_04q2`
- Web Server documentation:
  `http://docs.sun.com/coll/S1_websvr61_en`
- Application Server documentation
  `http://docs.sun.com/coll/s1_asseu3_en`
- Web Proxy Server documentation:
  `http://docs.sun.com/prod/s1.webproxys#hic`

Accessing Sun Resources Online

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

- Download Center:
  `http://wwws.sun.com/software/download/`
- Technical Support:
  `http://www.sun.com/service/support/software/`
- Sun Java Systems Services Suite:
  `http://www.sun.com/service/sunjavasystem/sjsservicessuite.html`
- Sun Enterprise Services, Solaris Patches, and Support:
  `http://sunsolve.sun.com/`

- Developer Information:
  http://developers.sun.com/prodtech/index.html

Contacting Sun Technical Support

If you have technical questions about this product that are not answered in the product documentation, go to:

http://www.sun.com/service/contacting.

Related Third-Party Web Site References

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 comments, 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 6 2005Q1 Federation Management Guide*, and the part number is 817-7648.

Part I

Liberty Specifications and Federation Management

Chapter 1

Introduction to the Liberty Alliance Project

Sun Java™ System Access Manager implements identity federation and Web services specifications defined by the Liberty Alliance Project. Before describing how this is accomplished, this appendix explains the concept of identity, identity services, the purpose of identity federation, and the role of the Liberty Alliance Project in creating identity-based solutions. It contains the following sections:

- Overview
- The Concept of Identity
- The Concept of Identity Federation
- Liberty Alliance Project Concepts
- The Liberty Alliance Project Specifications
- Deploying a Liberty-based System

Overview

In 2001 Sun Microsystems joined with other major companies to form the *Liberty Alliance Project* (LAP). The goal of the LAP is to define standards for developing identity-based infrastructures, software, and Web services, and to promote adoption of these standards. The LAP 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, feel free to move on to Chapter 2, “Implementation of the Liberty Specifications” which begins to describe how these standards are integrated into the Sun Java System Access Manager product.

LAP Members

The members of the LAP include some of the world’s most recognized brand names, representing products, services and partnerships across a wide spectrum of consumer and business service providers. The consortium also includes government organizations and technology vendors. A complete listing of current members of the LAP can be found at `http://www.projectliberty.org/membership/current_members.php`.

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

LAP Objectives

The specifications developed by the LAP enable individuals and organizations to securely conduct network transactions. More specifically, they:

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

The Concept of Identity

Identity can be defined as a set of information by which one person is definitively distinguished. In the real world, this information starts with a document that defines your name: a birth certificate. Over time, additional information further designates aspects of your 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 distinct documents represents data that defines your identity specifically to the enterprise for which it was issued. The composite of this data constitutes an overall identity with each specific piece detailing a distinguishing characteristic.

Because the Internet is becoming the primary vehicle for the interactions represented by this identity-defining information, people are now creating identities online for the enterprises with which they interact. By defining a user identifier and password, an email address, your personal preferences (style of music, access device, opt-in / opt-out marketing decision, email frequency), and other information more specific to the particular business (social security number, credit records, bank account number, bill payment information, ship-to address), users distinguish themselves from others who use the enterprise's services by creating this *virtual identity*. The virtual identity 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, it can make accessing each one time-consuming and frustrating. In addition, although

most local identities are configured independently (and fragmented across the Internet), it might be useful to connect the information; for example, your local identity with a bank could be securely connected to your local identity with a retailer. Identity federation is the solution to this issue.

The Concept of Identity Federation

Consider the many times you might access service provider accounts in a single day; sending and receiving email, logging in to a news portal, checking bank balances, finalizing travel arrangements, bidding on auction items, accessing utility accounts, and shopping online are all possible services for which you would define an identity. Each time you want to access one of these services, you identify yourself to the provider by logging in. If you use all of these services, you've configured a multitude of separate accounts that you must log in to (and log out of) for access. This virtual identity phenomenom offers the opportunity to fashion a system for computer users to link their local identities. *Identity federation* allows the user to associate, connect or bind the various local identities they have configured for multiple service providers. The linked local identities, referred to as a *federated identity*, then allow the user to log in to one service provider site and click through to an affiliated service provider site without having to re-authenticate or re-establish their identity. This notion of *single sign-on* is an option to which the user must agree. The *Liberty Alliance Project* was implemented to define standards using open technologies, therefore encouraging an interoperational infrastructure among service providers, and identity federation among users.

Liberty Alliance Project Concepts

A number of concepts are derived from the LAP specifications (discussed in "The Liberty Alliance Project Specifications" on page 35). Definitions for them are provided here.

Account Federation (Identity Federation)

Account federation occurs when a user chooses to unite distinct service provider accounts with one or more identity provider accounts. Users retain the individual account information with each provider while, simultaneously, establishing a link that allows the exchange of authentication information between them.

Affiliation

An *affiliation* is a group of providers formed without regard to their particular authentication domain. It is formed and maintained by an *affiliation owner*. An *affiliation document* describes a group of providers collectively identified by their `providerID`. 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).

Attribute Provider

An *attribute provider* is a web service that hosts attribute data. An example of an attribute provider would be an instance of the Personal Profile Service defined in “Liberty Identity Service Interface Specifications” on page 43.

Authentication Domain

A *authentication domain* is a group of service providers (with at least one identity provider) who agree to join together to exchange user authentication information using Liberty-enabled technologies. Once an authentication domain is established, single sign-on can be enabled amongst all membered providers. An authentication domain is sometimes referred to as a Circle Of Trust.

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

Circle Of Trust

See Authentication Domain.

Client

A *client* is actually the role any system entity assumes when making a request of another system entity. (In this scenario, the system entity of which the request is made is termed a Server.)

Common Domain

In an authentication domain having more than one identity provider, service providers need a way to determine which identity provider a principal uses. Because this function must work across any number of domain name system (DNS) domains, the Liberty approach is to create one domain common to all identity and service providers in the authentication domain. This predetermined domain is known as 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. Now, when the principal attempts to access another service provider within the authetnication domain, the service provider reads the common domain cookie and the request can be forwarded to the correct identity provider.

Defederation

See Federation Termination.

Federation Cookie

A *federation cookie* is a cookie implemented by Access Manager with the name `fedCookie`. It can have a value of either `yes` or `no` based on the principal's federation status. The concept was developed for Access Manager, and is not a defined part of the LAP specifications. Information on how a federation cookie is used can be found in "The Process of Federation" on page 62 of Chapter 3, "Federation Management."

Federated Identity

A *federated identity* refers to the amalgamation of the account information in all service providers accessed by one user (personal data, authentication information, buying habits and history, shopping preferences, etc.). The information is administered by the user yet, with the user's consent, privilege to access the information is securely shared with their providers of choice.

Federation Termination

Users have the ability to terminate their federations. *Federation termination* (or *defederation*) results in the cancellation of affiliations established between the user's identity provider and their federated service provider accounts.

Identity Provider

An *identity provider* is a service provider that specializes in providing authentication services. As the administrating service for authentication, identity providers also maintain and manage identity information. Authentication accomplished by an identity provider is honored by all service providers with whom it is affiliated. This term is used when defining an entity of this sort enabled by the ID-FF.

Identity Service

An *identity service* is a Web service that acts upon a resource to retrieve, update, or perform some action on data attributes related to a principal (an *identity*). An example of an identity service might be a corporate phone book or calendar service.

Liberty-enabled Client
A *Liberty-enabled client* is a client that has, or knows how to obtain, informatiuon 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. This pseudonym allows the providers to identify a principal without knowledge of the user's actual identity. The name identifier has meaning only in the context of the relationship between partners.

Principal
A *principal* is an entity that can acquire a federated identity, that is capable of making decisions, and to which authenticated actions are 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.

Pseudonym
See Name Identifier.

Receiver
A *receiver* is the role taken by 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 termed a Sender.)

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.

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 termed a Receiver.)

Server
A *server* is actually the role 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 termed a Client.)

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 enabled by the ID-FF.

Single Logout

A *single logout* occurs when a user logs out from an identity provider or a service provider. By logging out from one provider, they will effectively be logged out from 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. Because they have previously opted-in for federation, they are now able to access affiliated service providers without having to re-authenticate.

Trusted Provider

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

Web Service Consumer

A *Web service consumer* invokes the operations a Web service provides by making a request to a Web service provider. This term is used when defining an entity of this sort enabled by the ID-WSF.

Web Service Provider

A *Web service provider* implements a Web service based on a request from a Web service consumer. It may run on the same Java™ virtual machine as the Web service consumer using it. This term is used when defining an entity of this sort enabled by the ID-WSF.

The Liberty Alliance Project Specifications

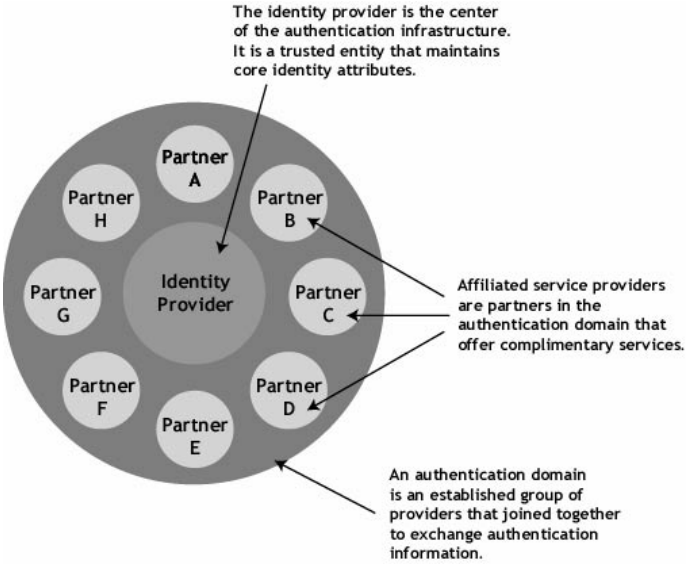
The LAP develops and delivers specifications that enable federated network identity management. Using Web redirection and open-source technologies like SOAP and XML, the LAP specifications enable distributed, cross-domain interactions. The LAP specifications are divided into three components:

- Liberty Identity Federation Framework
- Liberty Identity Web Services Framework
- Liberty Identity Service Interface Specifications

Liberty Identity Federation Framework

The *Liberty Identity Federation Framework* (ID-FF) defines a set of protocols, bindings and profiles that provides a solution for identity federation, cross-domain authentication and session management. These definitions can be used to create a brand new identity management system or develop one in conjunction with legacy systems. The ID-FF is designed to work with heterogeneous platforms, all types of networking devices (including personal computers, mobile phones, and PDAs), and other emerging technologies. A scenario implementing these specifications includes the subjects illustrated in Figure 0-1 and defined beneath it.

Figure 0-1 Concepts of the ID-FF Specifications



- A *principal* has a defined local identity with one or more providers, and has the option to federate these identities. The principal might be an individual user, a grouping 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 be it a news portal, a financial repository, or retail outlet. This broad category can also include:
  - utility companies
  - financial institutions
  - medical offices
  - corporate intranets
  - universities
  - government agencies
- An *identity provider* is a service provider that stores identity profiles and offers incentives to other service providers for the prerogative to federate their user identities. (Identity providers might also offer services above and beyond those related to identity profile storage.)
- In order to support identity federation, both service and identity providers must join together into an *authentication domain* (also referred to as a *circle of trust*). In an authentication domain, providers representing products, services and partnerships across a wide spectrum of consumer and business enterprises agree to join together to exchange authentication information using the LAP specifications. An authentication domain must contain at least one identity provider (to maintain and manage identity profiles) as well as at least two service providers. (One organization may be both an identity provider and a service provider.)

CAUTION In addition to integrating the LAP standards into their networks, organizations in an authentication domain must come to operational agreements to define their trust relationships. *Operational agreements* are a type of contractual relationship between organizations that defines how the domain will work. Operational agreements are out of the scope of the LAP specifications and this guide.

The set of ID-FF protocols include:

- Single Sign-on and Federation Protocol
- Name Registration Protocol

- Federation Termination Protocol
- Single Log-out Protocol
- Name Identifier Mapping Protocol
- Additional ID-FF Documents

NOTE More detailed information on the Liberty Identity Federation Framework can be found in the *Liberty ID-FF Protocols and Schema Specifications* (`http://www.projectliberty.org/specs/draft-liberty-idff-protocols-schema-1.2-errata-v1.0.pdf`).

Single Sign-on and Federation Protocol

The *Single Sign-on and Federation Protocol* defines a request / response protocol by which a principal is able to authenticate to a service provider, and *federate* (or link) their identities. A service provider issues a request for authentication to an identity provider. The identity provider responds with a message containing authentication information, or an *artifact* that points to authentication information which can then be de-referenced into authentication information. Additionally, the identity provider can 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. Principals can choose to federate their identity at the identity provider with their identity at the service provider.

Authentication context. Service providers can choose the type and level of authentication that should be used when the principal logs in.

Authentication credentials. Principals may be prompted for credentials at the behest of the service provider.

Account handle. An identity provider can issue an anonymous and temporary identifier to refer to a particular principal during communication with a service provider. This identifier is used to obtain information for or about principals (with their permission) during federation.

CAUTION This account handle is generated by the identity provider during federation unlike the handle that can be generated by the service provider after federation using the Name Registration Protocol.

Dynamic proxying. An identity provider that is asked to authenticate a principal it believes has already authenticated via another identity provider may make an authentication request on behalf of the requesting provider to the authenticating identity provider.

Identity provider introduction. When an authentication domain has more than one identity provider, a service provider can use this feature to discover 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 (HTTP, SOAP, etc.) used in the exchange are specified in *profiles*. Two of them are:

- The Liberty Artifact profile relies on SAML (Secure Access Markup Language) artifacts and assertions to relay authentication information.
- Liberty Browser POST profile relies on an HTML form to communicate authentication information between providers.

Name Registration Protocol

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

CAUTION This handle is not related to the opaque handle generated by the identity provider during federation as defined in the Single Sign-on and Federation Protocol. The Name Registration Protocol can, though, be used by the identity provider to change the opaque handle they registered with the service provider during initial federation.

Federation Termination Protocol

The *Federation Termination Protocol* defines how one provider (of either type) notifies another provider (of either type) when a principal has terminated their identity federation. The notification is in the form of a one-way, asynchronous message which states that either the service provider will no longer accept authentication information regarding the particular user, or the identity provider will no longer provide authentication information regarding the particular user.

Single Log-out Protocol

The *Single Log-out Protocol* defines how providers will notify each other of logout events. This message exchange protocol is used to terminate all sessions when a log out occurs at the service provider or identity provider level. The particular transfer and messaging protocol (HTTP, SOAP, etc.) used in the exchange are specified in *profiles*. Two of them 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 for a principal that has federated in the name space of a different service provider. This can be accomplished by querying an identity provider that has federated the user with both service providers. This allows the requesting provider to communicate with the other service provider without an identity federation for the principal between them.

Additional ID-FF Documents

Additional information explaining the ID-FF specifications can be found in the documents detailed in Table 0-1.

Table 0-1 Additional Help with the ID-FF

Name of Document	Overview
Liberty ID-FF 1.2 Architecture Overview `http://www.projectliberty.org/specs/liberty-idff-arch-overview-v1.2.pdf`	The Architecture Overview provides an architectural description of the ID--FF framework as well as policy, security and technical notes.
Liberty ID-FF 1.2 Protocols and Schema Specification `http://www.projectliberty.org/specs/draft-liberty-idff-protocols-schema-1.2-errata-v1.0.pdf`	The Protocols and Schema Specification provide the abstract Liberty protocols for Identity Federation, Single Sign-on, Name Registration, Federation Termination, and Single Log-out.
Liberty ID-FF 1.2 Implementation Guidelines `http://www.projectliberty.org/specs/liberty-idff-guidelines-v1.2.pdf`	The Implementation Guidelines provide guidance and checklists for implementing a Liberty-enabled environment using the ID-FF Specifications.

Table 0-1 Additional Help with the ID-FF

Name of Document	Overview
Liberty ID-FF 1.2 Static Conformance Requirements `http://www.projectliberty.org/specs/liberty-idff-1.2-scr-v1.0.pdf`	The Static Conformance Requirements define what features are mandatory and optional for implementations conforming to this version of the ID-FF Specifications.

Liberty Identity Web Services Framework

The 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* (ID-WSF) builds upon this by providing specifications to build Web services that retrieve, update, or perform an action on, identity data in a federated network environment. The specifications outline the technical components necessary to build Web services that interoperate with identity data, such as a calendar service, a wallet service, or an alert service. A scenario implementing these specifications includes the subjects defined below.

- A *Web service consumer* (WSC) invokes the operations 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 it provides. The WSP is the application implementing a Web service; it 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, it 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.

NOTE More information on the WSC/WSP process of the Liberty ID-WSF can be found in “Discovery Service Process” on page 125 of Chapter 6, “Discovery Service.”

The defined features of the ID-WSF include:

- SOAP Binding Specification
- Discovery Service Specification

- Security Mechanisms Specification
- Data Services Template Specification
- Interaction Service Specification
- Authentication Service Specification
- Client Profiles for Liberty-enabled User Agents or Devices
- Additional ID-WSF Documents

SOAP Binding Specification

The *SOAP Binding Specification* details a transport layer for handling SOAP messages. Among other features, it defines SOAP header blocks and processing rules enabling the invocation of identity services via SOAP requests and responses. It also specifies how to configure messages for optimum message correlation (assuring the relationship between a SOAP request and its response), consent claims (permission to perform a certain action), and usage directives (data handling policies).

Discovery Service Specification

The *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*, 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 it. A common use case is when a personal profile, or calendar data are placed within a discovery resource so that the data can be located by other entities.

Security Mechanisms Specification

The *Security Mechanisms Specification* describes the requirements for securing authorization decisions 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.

Data Services Template Specification

The *Data Services Template Specification* defines how to query and modify identity data attributes stored in a *data service* (a Web service that holds data). The specification also provides some common attributes for data services.

Interaction Service Specification

The *Interaction Service Specification* details 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 their identity data with requesting services.

Authentication Service Specification

The *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:

- 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 defined by the Discovery Service Specification.

Client Profiles for Liberty-enabled User Agents or Devices

The *Client Profiles for Liberty-enabled User Agents or Devices* describes the requirements for Liberty-enabled clients interacting with the SOAP-based Authentication Service. These profiles can enable browsers to perform an active role in transactions, in addition to the functions of a standard browser.

Additional ID-WSF Documents

Additional information explaining the ID-WSF specifications can be found in the documents detailed in Table 0-2 on page 43.

Table 0-2 Additional Help with the ID-WSF

Name of Document	Overview
Liberty ID-WSF Web Services Framework Overview `http://www.projectliberty.org/specs/liberty-idwsf-overview-v1.0.pdf`	The Web Services Framework Overview provides an architectural description of the ID-WSF framework including basic usage scenarios. It also highlights how the ID-WSF interacts with an identity management framework (such as the ID-FF).
Liberty ID-WSF Security and Privacy Overview `http://www.projectliberty.org/specs/liberty-idwsf-security-privacy-overview-v1.0.pdf`	The Security and Privacy Overview provides an overview of security and privacy issues in ID-WSF.

Liberty Identity Service Interface Specifications

The *Liberty Identity Service Interface Specifications* (ID-SIS) contain the following specifications for building these identity-based Web services:

- Personal Profile Service
- Employee Profile Service

Personal Profile Service

The *Personal Profile Service* defines an identity-based Web service that keeps, updates, and offers identity data regarding a user. The Personal Profile Service is characterized by the ability to query and update 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 the principal's account number and shopping preferences is an example of a personal profile service.

Employee Profile Service

The *Employee Profile Service* defines an identity-based web service which 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.

Supporting Documents

There are many other support documents in the LAP specifications. They include a metadata service protocol, reverse HTTP bindings, a glossary, and schema files. More information can be found at the LAP Web site or, more specifically, at http://www.projectliberty.org/resources/specifications.php#box4.

Deploying a Liberty-based System

This section details a few things to consider when building a successful Liberty-based implementation.

Size Up Your IT Staff

Although the LAP specifications are aimed at large organizations, small and medium-sized companies with a saavy IT staff can also roll out a federated identity system. The specifications are complicated and cross several domains of expertise (Web services development, XML, networking, and security).

Clean Your Directory Data

The LAP specifications do not specify where you store identity data; they are more concerned with it's accuracy. Purge your data store of old identity profiles, consolidate multiple (or delete duplicated) identity profiles, and ensure privileges are assigned correctly.

CAUTION Identity Providers should enforce strong passwords. A stolen identity can be abused across multiple sites in a federated system.

Draft Business Agreements

The LAP specifications assume pre-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 failing in them. When defined, a Liberty trust

relationship means that one organization trusts another's user authentication decisions. That trust among members lets a user log in at one site and access another site as well: single sign-on (SSO). Ensure that these agreements are in play before going live with a Liberty-compliant system.

Liberty-compliant Technology

At the minimum, a Liberty-compliant identity server is needed to process Liberty-based requests and responses. Chapter 2, "Implementation of the Liberty Specifications" begins our discussion of Sun Microsystems' implementation of the LAP specifications, the Sun Java™ System Access Manager.

Chapter 2

Implementation of the Liberty Specifications

Sun Java™ System Access Manager contains Sun Microsystems' implementation of the Liberty Alliance Project specifications. This chapter is an overview of how these specifications have been implemented. It contains the following sections:

- Overview
- Liberty Use Cases
- Access Manager Implementations
- Packages and Global Interfaces
- Liberty-based Samples

Overview

Sun Java System Access Manager is a software product that helps organizations manage secure access to the resources and Web applications both within the company and across the Internet. The initial releases of Access Manager (formerly Sun™ ONE Identity Server) implemented the Liberty Alliance Project (LAP) *Identity Federation Framework* (ID-FF) specifications, focusing on account federation, authentication domains and single sign-on.

NOTE The administration interface for managing the ID-FF implementation can be found in the Access Manager console by clicking the Federation Management tab in the Header frame.

Subsequently, Identity Server 2004Q2, added new features defined in version 1.2 of the ID-FF specifications as well as the version 1.0 specifications of the *Liberty Identity Web Services Framework* (ID-WSF). These Web services included a framework for the retrieval and update of *identity data* (attributes stored in identity-based Web services across the Internet), and a client application programming interface (API) for intracommunication between providers.

NOTE The Web interface for the ID-WSF implementation can be found in the Access Manager console by clicking the Service Management tab in the Header frame. Implemented Liberty-based services are listed amongst the other Web services.

This release, Sun Java System Access Manager 2005Q1, continues the implementation of Liberty-based features. The following sections detail features added to this latest version of Access Manager 2005Q1.

NOTE The full scope of Liberty Alliance Project features are discussed in Chapter 1, “Introduction to the Liberty Alliance Project.”

Name Identifier Mapping Protocol

The new Name Identifier Mapping Protocol, a full protocol in the *Liberty ID-FF Protocols and Schema Specifications* (http://www.projectliberty.org/specs/draft-liberty-idff-protocols-schema-1.2-errata-v1.0.pdf), allows a service provider to obtain a name identifier for a principal that has federated in the name space 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 NameIdentifier Mapping Profile can be found in the *Liberty ID-FF Bindings and Profiles Specification* (http://www.projectliberty.org/specs/draft-liberty-idff-bindings-profiles-1.2-errata-v1.0.pdf).

Single Sign-on and Federation Protocol

The following sections detail changes to the Single Sign-on and Federation Protocol, part of the *Liberty ID-FF Protocols and Schema Specifications* (http://www.projectliberty.org/specs/draft-liberty-idff-protocols-schema-1.2-errata-v1.0.pdf).

Dynamic Identity Provider Proxying

Dynamic Identity Provider Proxying can be enabled in an authentication request. For example, one identity provider might be asked to authenticate a principal that has already been authenticated via 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.

Affiliation Federation

Federation based on affiliation to a specified group can be enabled in an authentication request. If enabled, it would indicate that the requester is acting as a member of the affiliation group identified. Federations are then established and resolved based on the specified affiliation, and not the requesting provider. The process allows for a unique identifier that represents the affiliation.

One-Time Federation

The ability to federate for one session only can be enabled in an authentication request. This is useful for service providers with no user accounts, for principals who wish 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 Identifier Encryption Profile

The Name Identifier Encryption profile allows for 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 or multiple times, reducing the chance for correlation of the encrypted value across multiple logical transactions. The Name Identifier Encryption Profile can be found in the *Liberty ID-FF Bindings and Profiles Specification* (`http://www.projectliberty.org/specs/draft-liberty-idff-bindings-profiles-1.2-errata-v1.0.pdf`).

Liberty Metadata Description and Discovery Specification

The *Liberty Metadata Description and Discovery Specification*, one of the *Liberty Alliance Support Documents*, (`http://www.projectliberty.org/specs/draft-liberty-metadata-1.0-errata-v1.0.pdf`) has been upgraded to reflect added profiles, to support identity provider and service provider descriptors in the same metadata XML file, and to query metadata over the DNS.

NOTE Due to changes in the Liberty Metadata specification, the Service Management (SM) Configuration schema in Identity Server 6.2 is not compatible with that in Identity Server 6.1. SM versioning will be used to support coexistence of the two when running against an instance of Sun Java System Directory Server. When upgrading from Identity Server 6.1 to 6.2, metadata migration is required.

Liberty Use Cases

Identity data consists of all the information that companies capture and maintain about individual customers, corporate partners, and employees. Federating sources of identity data allows for accessing, transporting, sharing, and managing the data across and between partnered organizations and applications without weakening existing security safeguards. Federation management establishes this unifying network from multiple data stores. There are many ways to use Access Manager and its Liberty-based implementations to federate sources of identity data. The following sections detail just a few ways in which the product can be used.

Unified Access to Intranet Resources

Many corporations provide access to outsourced human resources services, such as health benefits and 401K 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 may not want to share the same profile and password with both their 401K provider and their healthcare provider. Federation of identity data can also provide seamless integration of Web resources across multiple security domains within the same enterprise making employee ease-of-use and control possible.

Integrated Partner Networks

Enterprises can construct a network of partnered services for securely exchanging customer account information, transaction data, and credentials via a set of interoperable Web services. Federating among partner networks allows identities to share key pieces of their respective data without sharing control. For example, logging onto 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.

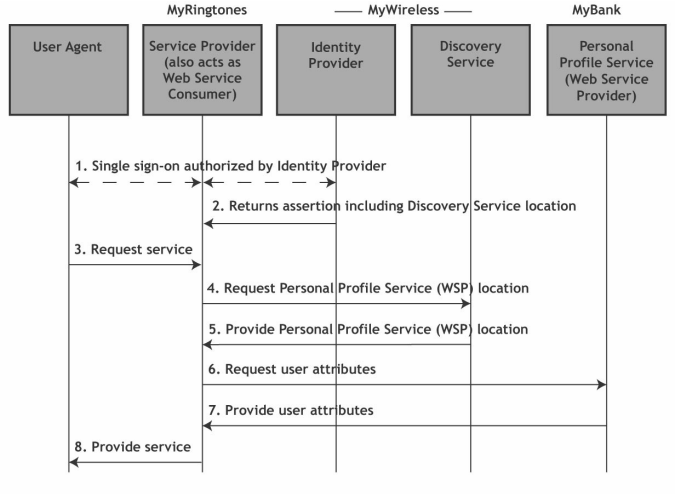
Sample Use Case Process

Figure 2-1 on page 52 illustrates the process of requesting a service and being authenticated for access. MyRingtones is a service provider in a federation framework that also acts as a Web service consumer in a Web services framework. MyWireless is an identity provider in a federation framework that contains access to the Discovery Service in a Web services framework. MyBank is a Web service provider in a Web services framework.

NOTE The same Web service can act as a different entity in different frameworks.

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, and the entire process is based on the implementations of the ID-FF, ID-WSF, and ID-SIS specifications of the LAP.

Figure 2-1 Process of Federation, Web Services & Service Instances Framework



Access Manager Implementations

Access Manager is installed with a set of default Liberty-based Web services. They, and the larger Federation Management module, are introduced in the following sections.

Web Services

Liberty-based Web services (based on the Liberty Identity Web Services Framework) are accessible from the Access Manager console by clicking the Service Management tab in the Header frame. Implemented services are listed alphabetically among other Access Manager Web services. Figure 2-2 is a screen shot of this.

Figure 2-2 Web Services Listed in Access Manager Console

Authentication Web Service

The Authentication Web Service provides Web service-based authentication to a Web service consumer (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 implementation of the Access Manager Authentication Web Service is based on the Liberty Alliance Project (LAP) “Authentication Service Specification.” The Access Manager Authentication Web Service is for service-to-service (non-user) authentication. More information can be found in Chapter 4, “Authentication Web Service.”

CAUTION The Liberty-based Authentication Web Service is not to be confused with the proprietary Access Manager Authentication Service discussed in the *Sun Java System Access Manager Developer's Guide* (`http://docs.sun.com/doc/817-5710`).

Discovery Service

The Discovery Service is an identity 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 describing the requested attribute provider. (A *resource offering* defines associations between a piece of identity data and the service instance that provides access to it.) The implementation of the Access Manager Discovery Service is based on the LAP "Discovery Service Specification" and includes Java and Web-based interfaces. More information can be found in Chapter 6, "Discovery Service."

NOTE By definition, a discoverable service is assigned a service type URI (typically done in the specification defining the service) allowing their registration in Discovery Service instances.

Liberty Personal Profile Service

The Liberty Personal Profile Service is an identity service that supports the storage and modification of identity data attributes regarding principals. Identity data 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. The implementation of the Access Manager Liberty Personal Profile Service is based on the LAP "Personal Profile Service." More information can be found in Chapter 5, "Data Services."

SOAP Binding

The SOAP Binding is a set of Java APIs used by the developer of a Liberty-enabled identity service that describes how to send and receive identity-based messages using SOAP, an XML-based messaging protocol. The implementation of the Access Manager SOAP Binding Service is based on the LAP "SOAP Binding Specification." More information can be found in Chapter 7, "SOAP Binding Service."

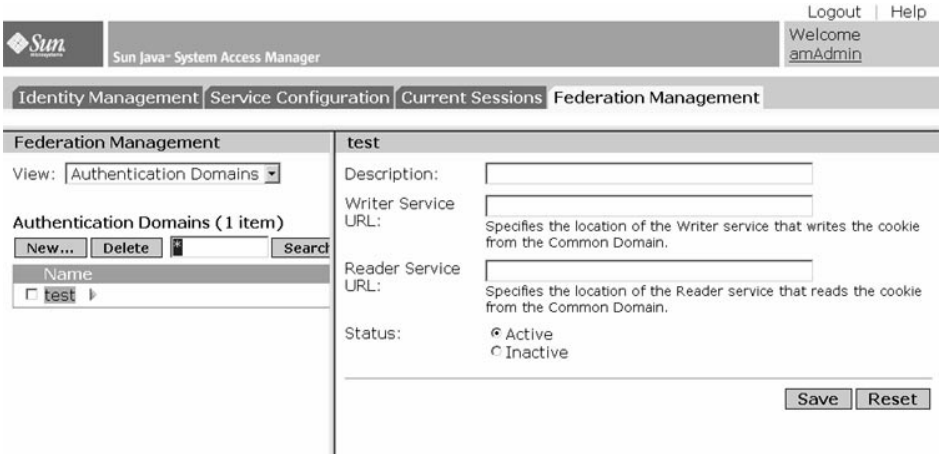
Application Programming Interfaces

A number of the Liberty-based Web services specifications have also been implemented in the back end of the Access Manager product as APIs. They include the interaction service, and PAOS binding. More information can be found in Chapter 8, “Application Programming Interfaces.”

Federation Management Module

The Federation Management module (based on the Liberty Identity Federation Framework) provides an interface for creating, modifying, and deleting authentication domains and, service and identity providers (both remote and hosted types) for a federated model. It is accessible through the Federation Management tab in the Header frame of the Access Manager console. Figure 2-3 on page 55 is a screen shot of this.

Figure 2-3 Federation Management Module in Access Manager Console



The following steps illustrate the basic procedure for creating a federation model.

1. Create an authentication domain.
2. Create one or more hosted providers that belong to the authentication domain.
3. Create one or more remote providers that belong to the authentication domain. You must also include the metadata for the remote providers.

4. Establish a trusted relationship 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.

NOTE The Federation Management module is the Web interface for the Access Manager implementation of the Liberty Identity Federation Framework.

Packages and Global Interfaces

Table 2-1 summarizes the public application programming interface (API) you can use to deploy Liberty-enabled components or extend the core services. For detailed API reference that includes classes, methods and their syntax and parameters, see the Javadocs in */AccessManager_base*/SUNWam/docs.

Table 2-1 Summary of Liberty-based Packages

Package Name	Description
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 155.
com.sun.identity.liberty.ws.common.wsse	Provides an interface to parse and create a X.509 Certificate Token Profile. See “Interaction Service API” on page 159.
com.sun.identity.liberty.ws.disco	Provides interfaces to manage the Discovery Service. See Chapter 6, “Discovery Service” on page 121.
com.sun.identity.liberty.ws.disco.plugins	Provides a plugin interface for the Discovery Service. See Chapter 6, “Discovery Service” on page 121.
com.sun.identity.liberty.ws.dst	Provides classes to implement an identity service on top of the Access Manager framework. The Data Services Template (DST) specification defines how to query and modify data stored in a data service, and provides some common attributes for the data services. From the implementation point of view, all the identity services must be built on top of the DST which provides the data model and message interfaces for all identity services. See “Interaction Service API” on page 159.
com.sun.identity.liberty.ws.interaction	Provides classes to support the Interaction RequestRedirect Profile. See “Interaction Service API” on page 159.
com.sun.identity.liberty.ws.interfaces	Provides interfaces common to all Access Manager Liberty-based Web service components. “Common Service Interfaces” on page 155.

Table 2-1 Summary of Liberty-based Packages

Package Name	Description
`com.sun.identity.liberty.ws.paos`	Provides classes for Web applications to construct and process PAOS requests and responses. See "PAOS Binding" on page 161 of Chapter 8, "Application Programming Interfaces."
`com.sun.identity.liberty.ws.security`	Provides interface to manage Liberty-based Web service security mechanisms. See "Common Security API" on page 157.
`com.sun.liberty`	Provides interfaces common to the Access Manager Federation Management module. See "Federation Management API" on page 87.

Liberty-based Samples

Access Manager has included sample code and files that can be used to understand the implementation of the LAP specifications. Information on the specifics of these samples can be found in Appendix A, "Included Samples."

Chapter 3

Federation Management

Sun Java™ System Access Manager provides an interface for creating, modifying, and deleting authentication domains and, service and identity providers (both remote and hosted types). This chapter is an overview of how to use this module to create a Liberty-based federation. It contains the following sections:

- Overview
- The Federation Management Interface
- The Process of Federation
- Common Domain Services
- Federation Management
- Federation Management API
- Federation Management Samples

Overview

The Federation Management module is the Access Manager implementation of the Liberty Alliance Project (LAP) *Liberty Identity Federation Framework* (ID-FF) specification. The ID-FF defines a set of protocols, bindings and profiles that provides a solution for identity profile federation, cross-domain authentication and session management. The Federation Management module is the Access Manager Web interface to the ID-FF implementation. It is accessible through the Federation Management tab in the Header frame of the Access Manager console.

NOTE More detailed information on the Liberty Identity Federation Framework can be found in the *Liberty ID-FF Protocols and Schema Specifications* (`http://www.projectliberty.org/specs/draft-liberty-idff-protocols-schema-1.2-errata-v1.0.pdf`).

The Federation Management Interface

The Federation Management module uses JavaServer Pages™ (JSP) to define its look and feel. JSP are HTML files that contain additional code to generate dynamic content. More specifically, JSP contain HTML code to display static text and graphics, as well as application code to generate information. When the page is displayed in a Web browser, it will contain both the static HTML content and, in the case of the Federation Management module, dynamic content retrieved via calls to the Federation Management API. An administrator can customize the look and feel of the interface by changing the HTML tags in the JSP, but the APIs invoked 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 Management module. To customize it for a specific organization, this default directory can be copied and renamed to reflect the name of the organization (or any value). It would then be placed at the same level as the default directory and the files within this directory would be modified as needed. Table 3-1 is a list of the JSP with details on what each page is used for and the invoked APIs that cannot be modified. More information on modifying these pages to customize the console can be found in the *Sun Java System Access Manager 6 2005Q1 Developer's Guide* (`http://docs.sun.com/doc/817-7649`).

Table 3-1 Federation Management Module JSP

File Name and its Purpose	Invoked APIs
CommonLogin.jsp displays the links to the login pages of the trusted identity providers as well as the local login page. It is displayed when the user is not logged in locally or at an identity provider site. The list of identity providers is obtained by the `getIDPList(hostedProviderID)` method.	• LibertyManager.getLoginURL(request) • LibertyManager.getInterSiteURL(request) • LibertyManager.getIDPList(providerID) • LibertyManager.getNewRequest(request) • LibertyManager.getSuccintID(idpID) • LibertyManager.cleanQueryString(request)
Error.jsp displays an error page when an error has occurred.	No APIs are invoked.

Table 3-1 Federation Management Module JSP *(Continued)*

File Name and its Purpose	Invoked APIs
Federate.jsp is displayed when the user clicks a Federate link. It displays a drop-down list of all providers with which the user is not yet federated. This list is constructed from the `getProvidersToFederate(userName, providerID)` method.	• LibertyManager.isLECPProfile(request) • LibertyManager.getAuthnRequestEnvelope(request) • LibertyManager.getUser(request) • LibertyManager.getProvidersToFederate(providerID,userDN)
FederationDone.jsp displays the status of federation (success or cancelled). It checks this status using the `isFederationCancelled(request)` method.	• LibertyManager.isFederationCancelled(request)
Footer.jsp displays a branded footer included on all the pages.	No APIs are invoked.
Header.jsp displays a branded header included on all the pages.	No APIs are invoked.
ListOfCOTs.jsp 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, they will be shown this JSP 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.	• LibertyManager.getListOfCOTs(providerID)
LogoutDone.jsp displays the status of the local logout operation.	• LibertyManager.isLogoutSuccess(request)
NameRegistration.jsp is displayed when a federated user chooses to register a new Name Identifier from a service provider to an identity provider. When the Name Registration link is clicked, this JSP is displayed.	• LibertyManager.getUser(request) • LibertyManager.getRegisteredProviders(userDN)
NameRegistrationDone.jsp displays the status of `NameRegistration.jsp`. When finished, this page is displayed.	• LibertyManager.isNameRegistrationSuccess(request) • LibertyManager.isNameRegistrationCanceled(request

Table 3-1 Federation Management Module JSP *(Continued)*

File Name and its Purpose	Invoked APIs
Termination.jsp is displayed when the user clicks the defederate link. It shows a drop-down menu of all providers to which the user has federated; from this list, the user can choose to defederate. The list is constructed using the `getFederatedProviders(userName)` method which returns all active providers to which the user is already federated.	• LibertyManager.getUser(request) • LibertyManager.getFederatedProviders(user DN)
TerminationDone.jsp displays the status of federation termination (success or cancelled). It checks status using the `isTerminationCancelled(request)` method.	• LibertyManager.isTerminationSuccess(request) • LibertyManager.isTerminationCanceled(request)

The Process of Federation

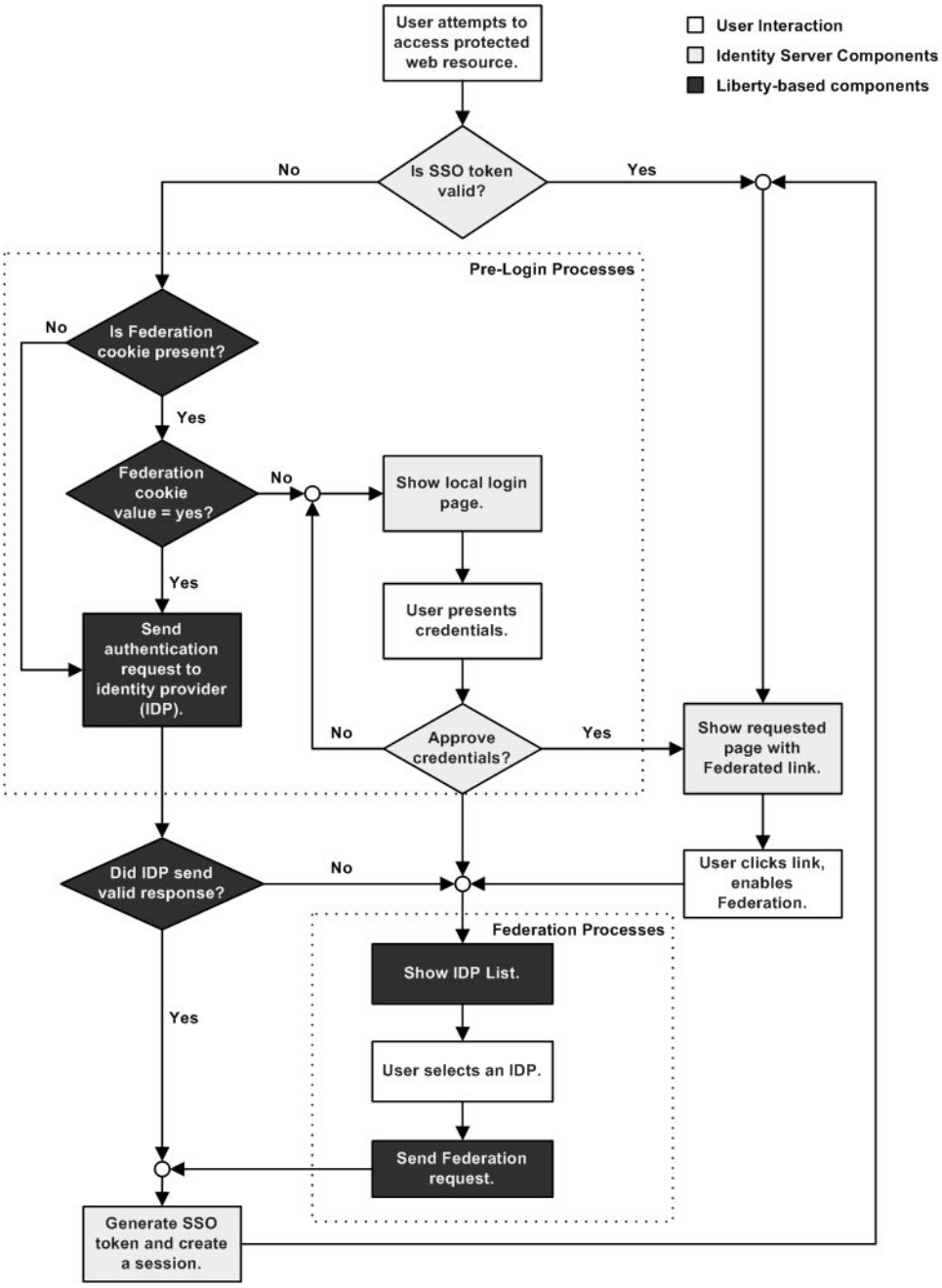
The process of federation begins with authentication. By default, Access Manager comes with two options for user authentication. The first is the proprietary Authentication Service; the second is the Liberty-enabled Federation process. With the proprietary option, users attempting to access a resource protected by Access Manager are redirected to the Authentication Service via an Access Manager login page. After they provide credentials, the Authentication Service allows or denies access to the resource based on the outcome.

NOTE For more information on the proprietary Authentication Service, see Chapter 4, Authentication Service in the *Sun Java System Access Manager 6 2005Q1 Developer's Guide* (`http://docs.sun.com/doc/817-7649`).

With Liberty-enabled federation, when a principal attempts to access a Web site belonging to a member provider from an authentication domain, the process begins with a search for a valid Access Manager session token from the Authentication Service. If a session token is found, the principal is granted (or denied) access. Assuming access is granted, the page then displayed would contain a link that provides the principal an opportunity to federate the authenticated identity provider identity with the accessed service provider identity. When the principal clicks this link, the Single Sign-on Process process begins.

If no session token is found, the principal is directed through the Pre-login Process. Figure 1-1 illustrates these different paths.

Figure 3-1 Liberty-based Access Manager Authentication Process Flow



Pre-login Process

The pre-login process establishes a valid session. When a principal attempts to access a service provider site and no Access Manager session token is found, the pre-login process then begins with the search for a *federation cookie*.

NOTE A *federation cookie* is a cookie implemented by Access Manager with the name `fedCookie`. It can have a value of either `yes` or `no` based on the principal's federation status. It is NOT detailed in the Liberty Alliance Project specifications.

The pre-login process can 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 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 user clicks this link, the Single Sign-on Process begins.
- 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 session. Authentication to Access Manager is accomplished 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 is sent to the principal's identity provider. (A passive authentication request does not allow identity provider interaction with the principal.) If an affirmative authentication is received back from the identity provider, the principal is redirected to the Access Manager Authentication Service a session token is granted and the principal is 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 the Liberty-enabled Single Sign-on Process.

Single Sign-on Process

When a principal logs in for access to 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 membered within the authentication domain. If the identity provider sends a negative response, the principal is directed to authenticate again using the Liberty-enabled single sign-on process.

In Liberty-enabled single sign-on, principals select an identity provider and send their credentials for authentication. (This is accomplished through the Common Domain Services.) Once authentication is complete and access is granted, the principal is automatically 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 sign on again.

Common Domain Services

The Common Domain Services allow a service provider to discover the specific identity provider used by a principal in an authentication domain with multiple identity providers. The Services rely on a cookie that is written in a domain that is common to all identity providers and service providers in the authentication domain. The domain (predetermined by all members of the authentication domain) is known as the *common domain*. The Common Domain Services use a *common domain cookie* (which contains a list of Base64-encoded identity provider identifiers) to determine the preferred identity provider.

NOTE The Common Domain Services are based on the Identity Provider Introduction Profiles detailed in the *Liberty ID-FF Bindings and Profiles Specifications* located at `http://www.projectliberty.org/specs/draft-liberty-idff-bindings-profiles-1.2-errata-v2.0.pdf`.

Let’s assume an authentication domain contains more than one identity provider. Because of this, 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 discover the correct one. When the request contains no common

domain cookie, the service provider presents a list of trusted identity providers from which the principal may choose. When the request contains a common domain cookie, the service provider reads the cookie to discover the correct identity provider.

Installing the Common Domain Services

The Common Domain Services for Federation Management are installed as one Web application within the Access Manager product using the Sun Java Enterprise System installer. However, they can also be installed as one Web application (separate from the Access Manager product) on a J2EE™ web container using the same installer.

NOTE For more information on installing the service, see the *Sun Java Enterprise System Installation Guide* on `docs.sun.com`. As of this writing, the latest version is available at `http://docs.sun.com/doc/817-5760`.

Common Domain Service URLs

In Access Manager, the Common Domain Services are exposed through two URLs that point to services developed for writing and reading the common domain cookie. The URLs are defined as attributes when an authentication domain is created.

NOTE The Reader and Writer service URLs are Access Manager specific. The concepts are not defined in the *Liberty ID-FF Bindings and Profiles Specifications*.

The format for the Writer Service URL is:

protocol: / / *common_domain_hostname*:*port* / *deloy_uri* / writer

The format for the Reader Service URL is:

protocol: / / *common_domain_hostname*:*port* / *deloy_uri* / transfer

See “To Create An Authentication Domain” on page 67 for information on configuring these attributes.

Federation Management

The Federation Management module in the Access Manager console provides an interface for creating, modifying, and deleting providers, authentication domains, and affiliations. The subsequent sections define these concepts and detail procedures for using the Federation Management interface.

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.

Authentication Domains

An *authentication domain* (also referred to as a *circle of trust*) 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 have established business relationships based on the LAP architecture and operational agreements.

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

Creating and Maintaining Authentication Domains

The following sections describe how to create, modify, and delete authentication domains using the Access Manager console.

To Create An Authentication Domain

1. Choose Authentication Domain from the View menu in the Navigation pane of the Federation Management module.
2. Click New in the Navigation pane.

   The New Authentication Domain attributes are displayed in the Data pane.
3. Enter a name for the authentication domain.

   This is a required field.

4. Enter a description of the authentication domain in the Description field.
5. Enter a value for the Writer Service URL.

   The Writer Service URL specifies the location of the service that writes the common domain cookie. The URL is in the format:

   `http://`*common_domain_host:port*`/common/writer`
6. Enter a value for the Reader Service URL.

   The Reader Service URL specifies the location of the service that reads the common domain cookie. The URL is in the format:

   `http://`*common_domain_host:port*`/common/transfer`
7. Select Active or Inactive.

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

   The new authentication domain is now displayed in the Navigation pane.

To Modify An Authentication Domain

1. Click on the Properties arrow next to the authentication domain you wish to modify in the Navigation pane of the Federation Management module.

   The authentication domain's properties are displayed in the Data pane.
2. Modify the properties of the authentication domain.
3. Click Save.

To Delete An Authentication Domain

1. Choose Authentication Domains from the View menu in the Navigation pane of the Federation Management module.

   All created Authentication Domains display in the Navigation pane.
2. Check the box next to the name of the Authentication Domain to be deleted.
3. Click Delete.

CAUTION Deleting an authentication domain does not delete the providers that belong to it.

Entity Descriptors

An *entity descriptor* contains one or more descriptions of individual providers, or affiliations. In the Access Manager Liberty implementation, there are two types:

- Provider Entity Descriptor
- Affiliate Entity Descriptor

Provider Entity Descriptor

The provider entity descriptor holds information configured for providers (both service and identity) associated with an authentication domain. Within this descriptor, the provider combinations detailed in Table 3-2 can be represented.

Table 3-2 Possible Provider Combinations for Provider Entity Descriptor

Entity	Description
Single Provider	This document defines one service or identity provider entity that can be referenced using a configured `providerID`.
Multiple Providers	This document combines multiple provider entities by referencing their configured `providerID`.

Affiliate Entity Descriptor

The affiliate entity descriptor holds information configured for a group of providers, but this group is formed outside of the boundaries of an authentication domain. This *affiliation* is formed and maintained by an *affiliation owner* that chooses trusted providers without regard to their particular authentication domain. This descriptor does not contain single or multiple providers unless they are specifically configured as an affiliation. An *affiliation document* describes a group of providers collectively identified by one `providerID` and maintained by an affiliation owner (referenced by its `affiliationOwnerID`). The document lists each member using their configured `providerID`.

NOTE More information on entity descriptors can be found in the *Liberty Metadata Description and Discovery Specification* (`http://www.projectliberty.org/specs/draft-liberty-metadata-1.0-errata-v2.0.pdf`).

Creating and Maintaining Entity Descriptors

Creating an entity descriptor using the Access Manager console is a two-step process. First, you create the entity descriptor itself. Then, you populate the descriptor with provider information (either service or identity) or an affiliation, depending on the descriptor created. The following sections describe how to create, modify, and delete entity descriptors using the Access Manager console.

To Create an Entity Descriptor of Either Type

1. Choose Entity Descriptors from the View menu in the Navigation pane of the Federation Management module.
2. Click New in the Navigation pane.

   The New Entity Descriptor attributes are displayed in the Data pane.
3. Enter a value for the Entity ID.

   This required field should specify the URL identifier of the entity. It must be unique across all entities.
4. Enter a description of the entity descriptor in the Description field.
5. Select Provider or Affiliate to define the Type.
   a. If you select Provider, click OK.
   b. If you select Affiliate, enter a value for both the Affiliate ID and Affiliate Owner ID attributes and click OK.

      The Affiliate ID should specify the URL identifier of the affiliate. It must be unique across all entities. The Affiliate Owner ID is the Provider ID of the owner or parent operator of the affiliation, from which additional metadata can be received. These fields are required.

   The new entity descriptor is now displayed in the Navigation pane.

To Configure a Provider Entity Descriptor

1. Choose Entity Descriptors from the View menu in the Navigation pane of the Federation Management module.
2. Select the desired provider entity descriptor.

   The entity descriptor's attributes are displayed in the Data pane.

To Configure General Attributes for a Provider Entity Descriptor

After selecting the desired provider entity descriptor from the Navigation pane:

1. Select General from the View menu in the Data pane and provide information for the following attributes (separated into three groups):

Entity Common Attributes

a. **Entity Type.** The static value of this attribute is Provider.

b. **Description.** Enter a description of the provider.

c. **Valid Until.** Enter the expiration date for the metadata pertaining to the provider. The value is defined in the format:

*yyyy-mm-dd*T*hh:mm:ss.SZ*

For example, `2004-12-31T12:30:00.0-0800`

d. **Cache Duration.** Enter the maximum amount of time the entity descriptor can be cached. The value is defined in 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.

Entity Contact Person

a. **First Name.** Enter the first name of the entity's contact person.

b. **Last Name.** Enter the last name of the entity's contact person.

c. **Type.** Select the type of entity from the drop down menu. The choices are Billing, Technical, Administrative, and Other.

d. **Company.** Enter the name of the company to which the contact person is employed.

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

f. **Email.** Enter the email address of the contact person.

g. **Telephone.** Enter the telephone number of the contact person.

Entity Organization

a. **Name.** Enter the name of the entity's organization. The value is defined in the format:

locale|*organization_name*

For example, `en`|*organization_name*`.com`

b. **Display Name.** Enter the display name of the entity's organization. The value is defined in the format:

locale|*organization_display_name*

For example, en|*organization_display_name*.com

c. **URL.** Enter the URL of the organization. The value is defined in the format:

locale|*organization_URL*

For example, en|http://www.*organization_name*.com

2. Click Save.

To Configure Identity Provider Attributes for a Provider Entity Descriptor

After selecting the desired provider entity descriptor from the Navigation pane:

1. Select Identity Provider from the View menu in the Data pane to add an identity provider to the entity descriptor.
2. Click the New Provider button to display the New Provider Wizard.
   a. Provide information for the following Common Provider attributes displayed in Step 1.
      I. **Provider ID.** Enter a unique identifier for the provider.
      II. **Description.** Enter a description of the provider.
      III. **Provider is Hosted or Remote.** Select Local if the provider is hosted on the same server as Access Manager or Remote, if not. By default, Remote is selected.

CAUTION Attributes displayed and configured in subsequent steps depend on the type defined for the Provider is Hosted or Remote attribute.

      IV. **Valid Until.** Enter the expiration date for the metadata pertaining to the provider. The value is defined in the format:

*yyyy-mm-dd*T*hh:mm:ss.S*Z

For example, 2004-12-31T12:30:00.0-0800

      V. **Cache Duration.** Enter the maximum amount of time an entity descriptor can be cached. The value is defined in 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.

VI. **Protocol Support Enumeration.** Select the protocol release supported by this entity.

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

VII. **Server Name Identifier Mapping Binding.** Enter a URI describing the SAML authority binding used by the identity provider. Identifier mapping queries are able to locate and communicate with the SAML authority using this URI.

VIII. **Additional Meta Locations.** Enter the location of other relevant metadata concerning the provider.

Signing Key

I. **Key Alias.** Enter the signing certificate key alias used to sign requests and responses for a hosted (local) provider. For a remote provider, this is a public key that the provider uses to verify the signatures.

Encryption Key

I. **Key Alias.** Enter the security certificate alias. Certificates are stored in a JKS keystore file. Each specific certificate is mapped to an alias which is used to fetch the certificate.

II. **Key Size.** Enter the length for keys used by the Web service consumer when interacting with another entity.

III. **Encryption Method.** Choose the method of encryption. The choices are None, 3DES, AES, and DES.

b. Click Next to provide information for the following Communications and Service Provider attributes displayed in Step 2.

CAUTION Some of the following attribute subsections are displayed based upon whether the identity provider is defined as Remote or Hosted (Local) in Step III on page 72. This is called out in parentheses next to the heading.

Communication URLs

I. **SOAP Endpoint URL.** Enter a location for the identity provider's SOAP messages receiver.

This value communicates the location of the SOAP receiver in non-browser communications.

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

III. **Single Logout Service URL.** Enter a location to which service providers can send logout requests.

Single logout synchronizes the logout functionality across all sessions authenticated by the identity provider.

IV. **Single Logout Return URL.** Enter a location to which the identity provider will redirect the principal after completing a logout.

V. **Federation Termination Service URL.** Enter a location to which a service provider will send federation termination requests.

VI. **Federation Termination Return URL.** Enter a location to which the identity provider will redirect the principal after completing federation termination.

VII. **Name Registration Service URL.** Enter a location to which a service provider will send requests to specify the name identifier that will be used when communicating with the identity provider about a principal.

Registration can occur only after a federation session is established.

VIII. **Name Registration Return URL.** Enter a location to which the identity provider will redirect the principal after HTTP name registration has been completed.

IX. **Authentication Service URL.** Enter a location for the identity provider's ID-FF-based Authentication Service.

Communication Profiles

I. **Federation Termination Profile.** Select a profile to notify other providers of a principal's federation termination. The choices are SOAP and HTTP/Redirect.

II. **Single Logout Profile.** Select a profile to notify other providers of a principal's logout. The choices are SOAP and HTTP/Redirect.

III. **Name Registration Profile.** Select a profile to notify other providers of a principal's name registration. The choices are SOAP and HTTP/Redirect.

IV. **Single Sign-on/Federation Profile.** Select a profile used by a hosted provider for sending authentication requests. The choices are:

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

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

Proxy Authentication Configuration (only displayed when identity provider is defined as Remote)

I. **Enable Proxy Authentication.** If selected, this attribute enables proxy authentication for a service provider.

II. **Proxy Identity Providers List.** This attribute displays the list of identity providers that can be proxied for authentication.

III. **Maximum Number Proxies.** This attribute specifies the maximum number of identity provider proxies.

IV. **Use Introduction Cookie For Proxying.** If enabled, introductions will be used to find the proxying identity provider.

Access Manager Configuration (only displayed when identity provider is defined as Hosted (Local))

I. **Provider URL.** Enter the URL of the local identity provider.

II. **Alias.** Enter an alias name for the local identity provider.

III. **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).

IV. **Default Authentication Context.** Select the authentication context to be used if the identity provider does not receive it as part of a service provider request. It also specifies the authentication context used by the service provider when an unknown user tries to access a protected resource. The choices are Previous-Session, Time-Sync-Token, Smartcard, MobileUnregistered, Smartcard-PKI, MobileContract, Password, Password-ProtectedTransport, MobileDigitalID, and Software-PKI.

V. **Forced Authentication at Identity Provider.** Select the check box to indicate if the identity provider must reauthenticate (even during a live session) when an authentication request is received.

VI. **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

VII. **Organization DN.** Enter the location of the DN of the organization if each hosted provider chooses to manage users across different organizations leading to a hosted model.

VIII. **Liberty Version URI.** Enter the URI of the version of the Liberty specification.

IX. **Name Identifier Implementation.** This field allows the option for 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 to the service provider.

X. **Provider Home Page URL**. Enter the URL of the home page of the identity provider.

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

SAML Configuration (only displayed when identity provider is defined as Hosted (Local))

I. **Assertion Interval.** Enter the interval of time for which an assertion issued by the identity provider will remain valid. A principal will remain authenticated until the assertion interval expires.

II. **Cleanup Interval**. Enter the interval of time before assertions stored in the identity provider will be cleared.

III. **Artifact Timeout.** Enter an interval to specify the timeout of a identity provider for assertion artifacts.

IV. **Assertion Limit.** Enter a number to define the amount of assertions an identity provider can issue, or the number of assertions that can be stored.

c. Click Next to provide information for the following Organization Attributes and Contact Persons attributes displayed in Step 3.

Organization

I. **Name.** Enter the name of the entity's organization. The value is defined in the format:

locale|*organization_name*

For example, en|*organization_name*.com

II. **Display Name.** Enter the display name of the entity's organization. The value is defined in the format:

locale|*organization_display_name*

For example, en|*organization_display_name*.com

III. **URL.** Enter the URL of the organization. The value is defined in the format:

locale|*organization_URL*

For example, en|http://www.*organization_name*.com

d. Click New to access the attributes for Contact Persons.

Contact Persons

I. **First Name.** Enter the first name of the entity's contact person.

II. **Last Name.** Enter the last name of the entity's contact person.

III. **Type.** Select the type of entity from the drop down menu. The choices are Billing, Technical, Administrative, and Other.

IV. **Company.** Enter the name of the company to which the contact person is employed.

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

VI. **Email.** Enter the email address of the contact person.

VII. **Telephone.** Enter the telephone number of the contact person.

e. Click OK to save the values assigned to the Contact Person attributes.

f. Click Next to configure the Authentication Domains to which the provider belongs in Step 4.

I. Use the direction arrows to move a Selected authentication domain into the Available list.

II. Click Save.

This will assign the provider to an authentication domain. 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.

g. Click Finish.

To Configure Service Provider Attributes for a Provider Entity Descriptor

After selecting the desired provider entity descriptor from the Navigation pane:

1. Select Service Provider from the View menu to add a service provider to the entity descriptor.
2. Click the New Provider button to display the New Provider Wizard.
   a. Provide information for the following Common Provider attributes displayed in Step 1.
      I. **Provider ID.** Enter a unique identifier for the provider.
      II. **Description.** Enter a description of the provider.
      III. **Provider is Hosted or Remote.** Select Local if the provider is hosted on the same server as Access Manager or Remote, if not. By default, Remote is selected.

CAUTION Attributes displayed and configured in subsequent steps depend on the type defined for the Provider is Hosted or Remote attribute.

      IV. **Valid Until.** Enter the expiration date for the metadata pertaining to the provider. The value is defined in the format:

      *yyyy-mm-dd*T*hh:mm:ss.S*Z

      For example, `2004-12-31T12:30:00.0-0800`

      V. **Cache Duration.** Enter the maximum amount of time an entity descriptor can be cached. The value is defined in 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.

      VI. **Protocol Support Enumeration.** Select the protocol release supported by this entity.

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

VII. **Server Name Identifier Mapping Binding.** Enter a URI describing the SAML authority binding used by the identity provider. Identifier mapping queries are able to locate and communicate with the SAML authority using this URI.

VIII. **Additional Meta Locations.** Enter the location of other relevant metadata concerning the provider.

Signing Key

I. **Key Alias.** Enter the signing certificate key alias used to sign requests and responses for a hosted (local) provider. For a remote provider, this is a public key that the provider uses to verify the signatures.

Encryption Key

I. **Key Alias.** Enter the security certificate alias. Certificates are stored in a JKS keystore file. Each specific certificate is mapped to an alias which is used to fetch the certificate.

II. **Key Size.** Enter the length for keys used by the Web service consumer when interacting with another entity.

III. **Encryption Method.** This field defines the encryption method. The choices are None, 3DES, AES, and DES.

b. Click Next to provide information for the following Communications and Service Provider attributes in Step 2.

CAUTION Some of the following attribute subsections are displayed based upon whether the service provider is defined as Remote or Hosted (Local) in Step III on page 78. This is called out in parentheses next to the heading.

Communication URLs

I. **SOAP Endpoint URL.** Enter a location for the service provider's SOAP messages receiver.

This value communicates the location of the SOAP receiver in non-browser communications.

II. **Single Logout Service URL.** Enter a location to which service providers can send logout requests.

Single logout synchronizes the logout functionality across all sessions authenticated by the identity provider.

III. **Single Logout Return URL.** Enter a location to which the service provider will redirect the principal after completing a logout.

IV. **Federation Termination Service URL.** Enter a location to which a service provider will send federation termination requests.

V. **Federation Termination Return URL.** Enter a location to which the service provider will redirect the principal after completing federation termination.

VI. **Name Registration Service URL.** Enter a location to which a service provider will send requests to specify the name identifier that will be used when communicating with the identity provider about a principal.

Registration can occur only after a federation session is established.

VII. **Name Registration Return URL.** Enter a location to which the identity provider will redirect the principal after HTTP name registration has been completed.

VIII. **Authentication Service URL.** Enter a location for the identity provider's ID-FF-based Authentication Service.

Communication Profiles

I. **Federation Termination Profile.** Select a profile to notify other providers of a principal's federation termination. The choices are SOAP and HTTP/Redirect.

II. **Single Logout Profile.** Select a profile to notify other providers of a principal's logout. The choices are SOAP and HTTP/Redirect.

III. **Name Registration Profile.** Select a profile to notify other providers of a principal's name registration. The choices are SOAP and HTTP/Redirect.

IV. **Single Sign-on/Federation Profile.** Select a profile used by a hosted provider for sending authentication requests. The choices are:

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

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

Service Provider

I. **Assertion Consumer URL.** Enter the SAML endpoint to which a provider will send SAML assertions.

II. **Assertion Consumer Service URL ID.** Enter the identifier of the Assertion Consumer Service URL to be used as a reference in authentication requests.

This identifier is required if Protocol Support Enum (Step VI on page 78) is `urn:liberty:iff:2002-12`.

III. **Set Assertion Consumer Service URL as Default.** Select this check box to use the Assertion Consumer URL as the default.

IV. **Sign Authentication Request.** Select this check box to specify that the service provider send signed authentication and federation requests. The identity provider will not process unsigned requests.

V. **Name Registration After Federation.** Select this check box to allow for a service provider to participate in name registration after it has been federated. For more information, see "Name Registration Protocol" on page 38 of Chapter 1, "Introduction to the Liberty Alliance Project."

VI. **Name ID Policy.** Choose an option to determine the name identifier format generated by the identity provider. The choices are None, One-time, and Federated. This attribute value is part of the authentication request. If the Name ID Policy value is `federated`, the name identifier format is `urn:liberty:iff:2003:federated`.

VII. **Enable Affiliation Federation.** If enabled, federation based on affiliation IDs is allowed.

Access Manager Configuration (only displayed when service provider is defined as Hosted (Local))

I. **Provider URL.** Enter the URL of the local identity provider.

II. **Alias.** Enter an alias name for the local identity provider.

III. **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).

IV. **Default Authentication Context.** Select the authentication context to be used if the identity provider does not receive it as part of a service provider request. It also specifies the authentication context used by the service provider when an unknown user tries to access a protected resource. The choices are Previous-Session, Time-Sync-Token, Smartcard, MobileUnregistered, Smartcard-PKI, MobileContract, Password, Password-ProtectedTransport, MobileDigitalID, and Software-PKI.

V. **Forced Authentication at Identity Provider.** Select the check box to indicate if the identity provider must reauthenticate (even during a live session) when an authentication request is received.

VI. **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

VII. **Organization DN.** Enter the location of the DN of the organization if each hosted provider chooses to manage users across different organizations leading to a hosted model.

VIII. **Liberty Version URI.** Enter the URI of the version of the Liberty specification.

IX. **Name Identifier Implementation.** This field allows the option for 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 to the service provider.

X. **Provider Home Page URL**. Enter the URL of the home page of the identity provider.

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

SAML Configuration (only displayed when service provider is defined as Hosted (Local))

I. **Assertion Interval.** Enter the interval of time for which an assertion issued by the identity provider will remain valid. A principal will remain authenticated until the assertion interval expires.

II. **Cleanup Interval**. Enter the interval of time before assertions stored in the identity provider will be cleared.

III. **Artifact Timeout.** Enter an interval to specify the timeout of a identity provider for assertion artifacts.

IV. **Assertion Limit.** Enter a number to define the amount of assertions an identity provider can issue, or the number of assertions that can be stored.

Proxy Authentication Configuration

I. **Enable Proxy Authentication.** If selected, this attribute enables proxy authentication for a service provider.

II. **Proxy Identity Providers List.** This attribute displays the list of identity providers that can be proxied for authentication.

III. **Maximum Number Proxies.** This attribute specifies the maximum number of identity provider to be proxied.

IV. **Use Introduction Cookie For Proxying.** If enabled, introductions will be used to find the proxying identity provider.

c. Click Next to provide information for the following Organization Attributes and Contact Persons attributes displayed in Step 3.

Organization

I. **Name.** Enter the name of the entity's organization. The value is defined in the format:

locale|*organization_name*

For example, en|*organization_name*.com

II. **Display Name.** Enter the display name of the entity's organization. The value is defined in the format:

locale|*organization_display_name*

For example, en|*organization_display_name*.com

III. **URL.** Enter the URL of the organization. The value is defined in the format:

locale|*organization_URL*

For example, en|http://www.*organization_name*.com

d. Click New to access the attributes for Contact Persons detailed below.

Contact Persons

I. **First Name.** Enter the first name of the entity's contact person.

II. **Last Name.** Enter the last name of the entity's contact person.

III. **Type.** Select the type of entity from the drop down menu. The choices are Billing, Technical, Administrative, and Other.

IV. **Company.** Enter the name of the company to which the contact person is employed.

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

VI. **Email.** Enter the email address of the contact person.

VII. **Telephone.** Enter the telephone number of the contact person.

e. Click OK to save the values assigned to the Contact Person attributes.

f. Click Next to configure the Authentication Domains to which the provider belongs in Step 4.

I. Use the direction arrows to move a Selected authentication domain into the Available list.

II. Click Save.

This will assign the provider to an authentication domain. 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.

g. Click Finish.

To Configure an Affiliate Entity Descriptor

1. Choose Entity Descriptors from the View menu in the Navigation pane of the Federation Management module.
2. Select the desired affiliate entity descriptor.

   The entity descriptor's attributes are displayed in the Data pane.

To Configure General Attributes for an Affiliate Entity Descriptor

After selecting the desired affiliate entity descriptor from the Navigation pane:

1. Select General from the View menu in the Data pane and provide information for the following attributes (separated into three groups):

   Entity Common Attributes

   a. **Entity Type.** The static value of this attribute is Affiliate.

   b. **Description.** Enter a description of the affiliation.

c. **Valid Until.** Enter the expiration date for the metadata pertaining to the affiliation. The value is defined in the format:

   *yyyy-mm-dd*T*hh:mm:ss.SZ*

   For example, `2004-12-31T12:30:00.0-0800`

d. **Cache Duration.** Enter the maximum amount of time the entity descriptor can be cached. The value is defined in 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.

Entity Contact Person

a. **First Name.** Enter the first name of the entity's contact person.

b. **Last Name.** Enter the last name of the entity's contact person.

c. **Type.** Select the type of entity from the drop down menu. The choices are Billing, Technical, Administrative, and Other.

d. **Company.** Enter the name of the company to which the contact person is employed.

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

f. **Email.** Enter the email address of the contact person.

g. **Telephone.** Enter the telephone number of the contact person.

Entity Organization

a. **Name.** Enter the name of the entity's organization. The value is defined in the format:

   locale|*organization_name*

   For example, `en`|*organization_name*`.com`

b. **Display Name.** Enter the display name of the entity's organization. The value is defined in the format:

   locale|*organization_display_name*

   For example, `en`|*organization_display_name*`.com`

c. **URL.** Enter the URL of the organization. The value is defined in the format:

locale|*organization_URL*

For example, `en|http://www.`*organization_name*`.com`

2. Click Save.

To Configure Affiliates Attributes for an Affiliate Entity Descriptor

After selecting the desired affiliate entity descriptor from the Navigation pane:

1. Select Affiliates from the View menu in the Navigation pane.
2. Provide information for the following Affiliate Common attributes (separated into three groups):

Affiliate Common Attributes

a. **Affiliate ID.** The value of this attribute should be defined during the creation of the Affiliate Entity Descriptor. For more information, see "To Create an Entity Descriptor of Either Type" on page 70.

b. **Affiliate Owner ID.** The value of this attribute should be defined during the creation of the Affiliate Entity Descriptor. For more information, see "To Create an Entity Descriptor of Either Type" on page 70.

c. **Valid Until.** Enter the expiration date for the metadata pertaining to the provider. The value is defined in the format:

*yyyy-mm-dd*T*hh:mm:ss.S*Z

For example, `2004-12-31T12:30:00.0-0800`

d. **Cache Duration.** Enter the maximum amount of time an entity descriptor can be cached. The value is defined in 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

a. **Key Alias.** Enter the signing certificate key alias used to sign requests and responses for a hosted (local) provider. For a remote provider, this is a public key that the provider uses to verify the signatures.

Encryption Key

a. **Key Alias.** Enter the security certificate alias. Certificates are stored in a JKS keystore file. Each specific certificate is mapped to an alias which is used to fetch the certificate.

b. **Key Size.** Enter the length for keys used by the Web service consumer when interacting with another entity.

c. **Encryption Method.** Choose the method of encryption. The choices are None, 3DES, AES, and DES.

Affiliate Members

a. **Affiliate Members.** Use the direction arrows to move a Selected provider into the Available list.

This field allows you to define one or more providers as members of the affiliation. The providers displayed in the Selected list are pre-defined in existing container entity descriptors.

3. Click Save.

To Delete an Entity Descriptor of Either Type

1. Choose Entity Descriptors from the View menu in the Navigation pane of the Federation Management module.
2. Check the box next to the entity descriptor you want to delete.
3. Click Delete.

There is no warning message when performing a delete.

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

Federation Management API

The `com.sun.liberty` package provides the interface that forms the basis of the Federation Management API. The `LibertyManager` class must be instantiated by web applications that want to access the Federation Management module. It contains the methods needed by the module JSPs for account federation, session termination, log in, log out and other actions. Some of these methods are:

Table 3-3 Federation Management API

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

For more detailed API reference information, see the Javadocs in */AccessManager_base*/SUNWam/docs.

Federation Management Samples

Access Manager provides a collection of sample files, located in the */AccessManager_base*/SUNWam/samples/liberty/Sample1 directory, to configure a basic environment for creating and managing a federation. The example demonstrates the basic use of various Liberty-based federation protocols including account federation, SSO, single logout, and federation termination. The sample should be completed in the following sequence:

1. Install Access Manager
2. Update and load the metadata
3. Deploy the service provider

4. Deploy the identity provider
5. Create and manage the federation

The following sections include more information on these steps.

NOTE The Readme file located with the sample in */AM_Install_Dir*/SUNWam/samples/liberty/sample1 also contains instructions for configuring a common domain. For information on common domains, see "Common Domain" on page 31 of Chapter 1, "Introduction to the Liberty Alliance Project" and "Common Domain Services" on page 65 of this chapter.

Installing Access Manager

The first step in creating a federated environment is installing Access Manager on two separate machines. One installation will act as a service provider, and one will act as an identity provider.

NOTE Instructions on installing Access Manager can be found in the *Sun Java Enterprise System Installation Guide* (http://docs.sun.com/coll/entsys_05q1).

The default installation directory for the Solaris™ operating system is /opt/SUNWam.

Updating and Loading the Metadata

Update and load the sp1Metadata.xml file with values appropriate to your Access Manager installation. The file is located in */AccessManager_base*/SUNWam/samples/liberty/sample1. Table 3-4 summarizes the default values which should be modified based on your installation configuration.

Table 3-4 Default Values in sp1metadata.xml for Sample1

Installation Parameter	Service Provider Value	Identity Provider Value
Provider Name	SP1	IDP1
Host Name	www.sp1.com	www.idp1.com
Port	*SERVER_PORT_#*	*SERVER_PORT_#*
Access Manager Deployment URI	amserver	amserver

Table 3-4 Default Values in sp1metadata.xml for Sample1

Installation Parameter	Service Provider Value	Identity Provider Value
Access Manager root suffix	dc=sp1,dc=com (attribute DN for element OrganizationRequests)	dc=idp1,dc=com (attribute DN for element OrganizationRequests)
Certificate Alias	SP1_SECURITY_KEY	IDP1_SECURITY_KEY
metaAlias	www.sp1.com	www.idp1.com

Load the updated sp1Metadata.xml file using the following command:

/AccessManager_base/SUNWam/bin/amadmin -u amadmin -w *password* -t sp1Metadata.xml

Deploying the Service Provider

The following sequence should be followed in order to deploy the service provider:

1. Configure the AMClient.properties file.
2. Create a WAR file.
3. Deploy the WAR file.

To Configure AMClient.properties

Replace the following tags in the AMClient.properties file with values appropriate to your configuration. AMClient.properties is located in */AccessManager_base*/SUNWam/samples/liberty/sample1/sp1/WEB-INF/classes/.

- **SERVER_PROTO:** Enter HTTPS or HTTP.
- **SERVER_HOST:** Enter the fully-qualified host name for your installation. For example, www.sp1.com.
- **SERVER_PORT:** Enter the port number on which Access Manager is running.
- **SERVICE_DEPLOY_URI:** Enter the Access Manager services deployment URI. The default value is amserver.
- **META_ALIAS:** Enter the metaAlias for SP1. In sp1Metadata.xml, the default value is www.sp1.com.

To Create a WAR File for SP1

1. Change to the `sp1` directory.

   ```
   cd /AccessManager_base/SUNWam/samples/liberty/sample1/sp1
   ```

2. Run the `jar` command.

   ```
   jar -cvf sp1.war
   ```

To Deploy the Service Provider WAR File

Choose the option appropriate to your environment.

- If Access Manager is Installed on Sun Java System Web Server
- If Access Manager is Installed on Sun Java System Application Server

NOTE Instructions for deploying the WAR file on other application servers can be found in the Readme file located with the sample in */AM_Install_Dir*`/SUNWam/samples/liberty/sample1`.

If Access Manager is Installed on Sun Java System Web Server

CAUTION Before manually deploying a web application, be sure that the:

- *server_root*`/bin/https/httpsadmin/bin` directory is in your path.
- `IWS_SERVER_HOME` environment variable is set to your `server_root` directory.

1. Enter the command
   `wdeploy deploy -u` *uri_path* `-i` *instance* `-v` *vs_id* `[-d` *directory*`]` *war_file*

   where:

   - *uri_path* is the URI prefix for the web application.
   - *instance* is the server instance name.
   - *vs_id* is the virtual server ID.
   - *directory* is the directory to which the application is deployed. If not specified, the application is deployed to the document root directory.

- *war_file* is the WAR file name.

An example might be:

```
wdeploy deploy -u /sp1 -i www.sp1.com -v https-www.sp1.com
    -d begin_dir/web-apps/sp1 sp1.war
```

2. Restart the Web Server.

If Access Manager is Installed on Sun Java System Application Server

1. Use the `asadmin deploy` command to deploy the WAR module.

The complete syntax is:

```
asadmin deploy --user admin_user [--password admin_password]
  [--passwordfile password_file] --host hostname
    --port adminport [--secure | -s] [--virtualservers virtual_servers]
      --type application|ejb|web|connector]
        [--contextroot contextroot] [--force=true]
          [--precompilejsp=false] [--verify=false]
            [--name component_name] [--upload=true]
              [--retrieve local_dirpath]
                [--instance instance_name] path_to_file
```

For example:

```
asadmin deploy --user amadmin --password pswd1234
  --host www.sp1.com --port 4848 --type web --contextroot SP1
    --instance server1 sp1.war
```

2. Restart the Application Server.

Deploying the Identity Provider

The following sequence should be followed in order to deploy the identity provider:

1. Configure the `AMClient.properties` file.
2. Create a WAR file.
3. Deploy the WAR file.

To Configure AMClient.properties

Replace the following tags in the `AMClient.properties` file with values appropriate to your configuration. `AMClient.properties` is located in */AccessManager_base*`/SUNWam/samples/liberty/sample1/idp1/WEB-INF/classes/`.

- **SERVER_PROTO:** Enter HTTPS or HTTP.
- **SERVER_HOST:** Enter the fully-qualified host name for your installation. For example, `www.idp1.com`.
- **SERVER_PORT:** Enter the port number on which Access Manager is running.
- **SERVICE_DEPLOY_URI:** Enter the Access Manager services deployment URI. The default value is `amserver`.
- **META_ALIAS:** Enter the metaAlias for IDP1. In `idp1Metadata.xml`, the default value is `www.idp1.com`.

To Create a WAR File for IDP1

1. Change to the `idp1` directory.

   `cd /`*AccessManager_base*`/SUNWam/samples/liberty/sample1/idp1`

2. Run the `jar` command.

   ```
   jar -cvf idp1.war
   ```

To Deploy the Identity Provider WAR File

Choose the option appropriate to your environment.

- If Access Manager is Installed on Sun Java System Web Server
- If Access Manager is Installed on Sun Java System Application Server

NOTE Instructions for deploying the WAR file on other application servers can be found in the Readme file located with the sample in *AM_Install_Dir*`/SUNWam/samples/liberty/sample1`.

If Access Manager is Installed on Sun Java System Web Server

CAUTION Before manually deploying a web application, be sure that the:

- *server_root*`/bin/https/httpsadmin/bin` directory is in your path.
- `IWS_SERVER_HOME` environment variable is set to your `server_root` directory.

1. Enter the command
   `wdeploy deploy -u` *uri_path* `-i` *instance* `-v` *vs_id* `[-d` *directory*`]` *war_file*

   where:

   - *uri_path* is the URI prefix for the web application.
   - *instance* is the server instance name.
   - *vs_id* is the virtual server ID.
   - *directory* is the directory to which the application is deployed. If not specified, the application is deployed to the document root directory.
   - *war_file* is the WAR file name.

   An example might be:

   ```
   wdeploy deploy -u /idp1 -i www.idp1.com -v https-www.idp1.com
       -d /AccessManager_base/SUNWam/web-apps/idp1 idp1.war
   ```

2. Restart the Web Server.

If Access Manager is Installed on Sun Java System Application Server

1. Use the `asadmin deploy` command to deploy the WAR module.

   The complete syntax is:

   ```
   asadmin deploy --user admin_user [--password admin_password]
     [--passwordfile password_file] --host hostname
       --port adminport [--secure | -s] [--virtualservers virtual_servers]
         --type application|ejb|web|connector]
           [--contextroot contextroot] [--force=true]
             [--precompilejsp=false] [--verify=false]
               [--name component_name] [--upload=true]
                 [--retrieve local_dirpath]
                   [--instance instance_name] path_to_file
   ```

   For example:

   ```
   asadmin deploy --user amadmin --password pswd1234
     --host www.idp1.com --port 4848 --type web --contextroot IDP1
       --instance server1 idp1.war
   ```

2. Restart the Application Server.

Creating and Managing a Federation

The following sections provide procedures for creating, managing, and terminating a federation.

- To Federate the Service Provider and Identity Provider Accounts
- To Accomplish Single Sign-On
- To Perform a Single Logout
- To Terminate Account Federation

To Federate the Service Provider and Identity Provider Accounts

1. Access the following URL in a web browser:

   SERVER_PROTO//*SERVER_HOST*:*PORT*/sp1/index.jsp

   For example, http://www.sp1.com:58080/sp1/index.jsp.

NOTE index.jsp is a protected page that includes _head.jsp. _head.jsp checks the request for a valid user session. If invalid, it redirects the request to the Pre-Login service which attempts single sign-on. Since this is a first time access, single sign-on will fail and the request is then redirected to the common login page.

2. Click the Local Login link on the common login page.

   You are redirected to the SP1's login page.

3. Log in to SP1.

   After successful authentication at SP1, the index.jsp is displayed. index.jsp has three links:

   - The Federate link initiates the federation process.
   - The Logout link initiates the single logout process.
   - The Terminate Federation link initiates the federation termination process.

4. Click the Federate link.

   The Federate page is displayed.

5. Select the identity provider with which you want to federate.

   In Sample1, you would select the deployed IDP1 as your identity provider, and IDP1's login page is displayed.

6. Provide authentication credentials for your IDP1 account.

   If the authentication is successful, the Federation Done page is displayed indicating that you have successfully federated these two accounts.

NOTE If the account is already federated, you will be redirected to the IDP login page

To Accomplish Single Sign-On

After successfully federating the two providers, follow these instructions to accomplish single sign-on.

1. Start a new browser session and access the SP1 protected page, *SERVER_PROTO*//*SERVER_HOST*:*PORT*/sp1/index.jsp.

   For example, http://www.sp1.com:58080/sp1/index.jsp.

2. You will be redirected to the IDP1 Login page for authentication.

3. Provide authentication credentials for your IDP1 account.

   If authentication is successful, the initially accessed SP1 protected page is displayed without asking for SP1 authentication credentials. If authentication is not successful, an error message is displayed, and you are directed to start over.

To Perform a Single Logout

From either the SP1 protected page or the IDP1 protected page, index.jsp, click the Logout link. You will be logged out from both providers, and the Logout Done page is displayed.

NOTE Both the service provider and identity provider have different protected index.jsp pages. The URLs are:

- *SERVER_PROTO*//*SERVER_HOST*:*PORT*/sp1/index.jsp
- *SERVER_PROTO*//*SERVER_HOST*:*PORT*/idp1/index.jsp

To Terminate Account Federation

1. From either the SP1 protected page or the IDP1 protected page, click the Terminate Federation link.

   The Federation Termination page is displayed.

2. Select a provider to terminate your account federation.

   For Sample1, select IDP1. Upon successful federation termination, the Termination Done page is displayed.

NOTE Appendix A, “Included Samples” includes information on two more samples that make use of the Federation Management module.

Part II

Liberty-based Web Services

Chapter 4

Authentication Web Service

The Sun Java™ System Access Manager contains an implementation of the *Liberty ID-WSF Authentication Service Specification* of the Liberty Alliance Project. The Authentication Web Service defines how to perform authentication using SOAP. This chapter contains the following topics:

- Overview
- Authentication Web Service Process
- Authentication Web Service Attribute
- Authentication Web Service Interfaces
- Authentication Web Service Sample

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 7, "SOAP Binding Service.") The Simple Authentication and Security Layer (SASL) is the method used to add this authentication support to the SOAP transport layer. The Access Manager Authentication Web Service is for service-to-service (non-user) authentication.

NOTE On the Liberty Alliance Project Web site, the *Liberty ID-WSF Authentication Service Specification* can be found at `http://www.projectliberty.org/specs/liberty-idwsf-authn-svc-v1.0.pdf`.

XML Service File

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

NOTE More information on XML service files can be found in the section on XML Service Files in the *Sun Java System Access Manager 6 2005Q1 Developer's Guide* (http://docs.sun.com/doc/817-7649).

The *Liberty ID-WSF Authentication Service Specification* also contains an XML schema that defines the authentication protocol. This XML Schema Defintion (XSD) file can be found on the LAP Web site. Version 1.0 is also reproduced in Appendix B, "Service Schema Files."

Application Programming Interfaces

The Access Manager Authentication Web Service includes two Java programming packages: com.sun.identity.liberty.ws.authnsvc.protocol and com.sun.identity.liberty.ws.authnsvc. The former listed package contains classes that represent the SASL request and response while the latter package is a client API for external Java applications to send SASL requests and receive SASL responses. They are used to initiate the authentication process and communicate authentication credentials to the Authentication Web 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).

NOTE The authentication exchange can involve an arbitrary number of round trips, dictated by the particular SASL mechanism employed. The WSC may have knowledge of the supported SASL mechanisms, or it may send the server its own list and allow the server to choose one from among them. 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 following steps detail the sequence between the WSC and the Authentication Web Service (a WSP).

1. The authentication exchange begins with a WSC sending 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 disproving the authentication.

   If approved, the response includes the credentials the WSC needs to invoke other Web services (like the Discovery Service).

CAUTION The Liberty-based Authentication Web Service is not to be confused with the proprietary Access Manager Authentication Service discussed in the *Sun Java System Access Manager 6 2005Q1 Developer's Guide* (`http://docs.sun.com/doc/817-7649`).

Authentication Web Service Attribute

The Authentication Web Service attribute is a *global* attribute. The value of this attribute is carried across the Sun Java System Access Manager configuration and inherited by every organization.

NOTE For information on the types of attributes used in Access Manager, see the Service Management chapter of the *Sun Java System Access Manager 6 2005Q1 Developer's Guide* (`http://docs.sun.com/doc/817-7649`).

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

Mechanism Handler List

The Mechanism Handler List attribute stores information about the SASL mechanisms supported by the Authentication Web Service. It displays entries that contain key / value pairs separated by a pipe (“|”) as in:

```
key=PLAIN|class=com.sun.identity.liberty.ws.authnsvc.mechanism.PlainMechanismHandler
```

key Parameter

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

class Parameter

The required *class* parameter specifies the name of the implementation class for the SASL mechanism. The Authentication Web Service layer provides a handler interface that needs to be implemented in order for each SASL mechanism to process the requested message and return a response.

Authentication Web Service Interfaces

The Authentication Web Service provides programmatic interfaces to allow clients to interact with the Authentication Web Service. They are:

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

com.sun.identity.liberty.ws.authnsvc

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.protocol

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 reprod

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 SAML *Bearer* token credential, issues a discovery query request with SAML assertion included, and gets back a response.

NOTE This sample can be used a Liberty User Agent Device WSC.

Chapter 5

Data Services

The Sun Java™ System Access Manager contains implementations of the *Liberty ID-WSF Data Services Template Specification* (ID-WSF-DST) in addition to instructions on how you can add your own data service to the deployment. This chapter contains the following topics:

- Overview
- Liberty Personal Profile Service
- Liberty Employee Profile Service
- Data Services Template API
- Developing A New Data Service

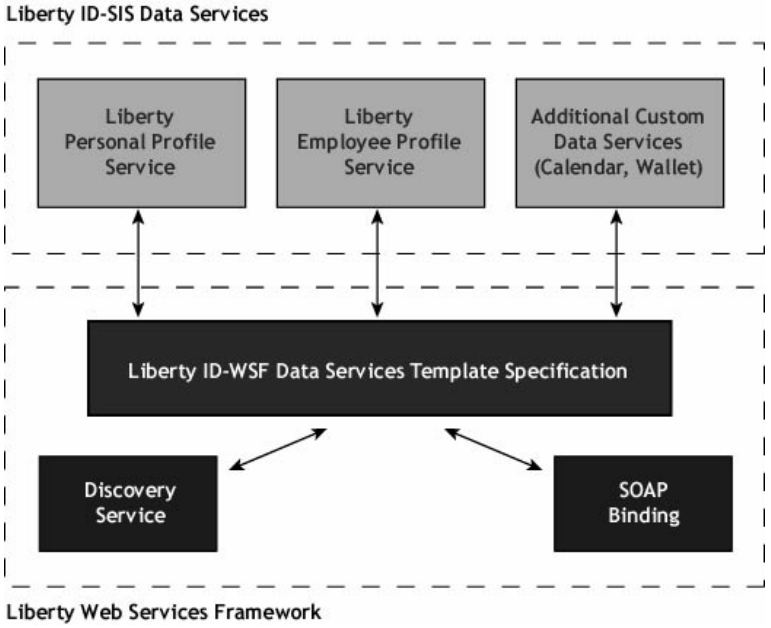
Overview

A *data service* is a Web service that supports the query and modification of identity data. *Identity data* includes, but is not limited to, attributes that define first name, last name, home address, business address, and emergency contact. A data service allows this data to be queried or modified. A *query* is when a Web service consumer (WSC) requests data (in an XML format) from a user's profile. A *modify* is when a WSC sends new data to update a user's profile. The Liberty Alliance Project (LAP) has defined the *Liberty ID-WSF Data Services Template Specification* (ID-WSF-DST) as the standard protocol used for the query and modification of identity data profiles comprised of attributes exposed by a data service.

Data Services Template Specifications

The 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 their identity. The information may include the individual's office location and phone number, as well as job title or department name. From the implementation point of view, all data services must be built on top of the ID-WSF-DST which provides the data model and message interfaces. Figure 5-1 illustrates how Access Manager uses the ID-WSF-DST as the framework for its data services.

Figure 5-1 Data Service Template as Building Block for Data Services



The Liberty-defined Web Services Layer uses the ID-WSF-DST (and other Web services that allow data services to be discovered and invoked) for the development of data services. Access Manager has developed both the Liberty Personal Profile Service and the Liberty Employee Profile Service on top of the Liberty-defined Web Services Layer. Additional data services can also be developed by the customer. (More information on developing other data services can be found in "Data Services Template API" on page 118.)

NOTE The *Liberty ID-WSF Data Services Template Specification* can be found at http://www.projectliberty.org/specs/draft-liberty-idwsf-dst-1.0-errata-v1.0.pdf.

Liberty Personal Profile Service

The *Liberty ID-SIS Personal Profile Service Specification* (ID-SIS-PP) of the LAP describes a data service which provides an identity's basic profile information (full name, contact details, financials, etc.). It 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.

XML Service File

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

NOTE More information on XML service files can be found in the section on XML Service Files in the *Sun Java System Access Manager 6 2005Q1 Developer's Guide* (http://docs.sun.com/doc/817-7649).

XSD Schema Definition

The ID-SIS-PP also defines an XML schema for use in building the service itself. This XML Schema Defintion (XSD) file can be found on the LAP Web site. Version 1.0 is also reproduced in Appendix B, "Service Schema Files."

NOTE The *Liberty ID-SIS Personal Profile Service Specification* can be found at http://www.projectliberty.org/specs/liberty-idsis-pp-v1.0.pdf.

Liberty Employee Profile Service

The *Liberty ID-SIS Employee Profile Service Specification* (ID-SIS-EP) describes a data service which provides an identity's profile information in regards to their 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.

TIP The Liberty Employee Profile Service is not available when Access Manager is installed. It must first be deployed. Information on accessing the sample files and how to deploy them can be found in "Liberty Employee Profile Service" on page 118.

XML Service File

Among the files included with the sample is the XML service file `amLibertyEmployeeProfile.xml`. `amLibertyEmployeeProfile.xml` 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 More information on XML service files can be found in the section on XML Service Files in the *Sun Java System Access Manager 6 2005Q1 Developer's Guide* (`http://docs.sun.com/doc/817-7649`).

XSD Schema Definition

The ID-SIS-EP also defines an XML schema for use in building the service itself. This XSD file can be found on the LAP Web site. Version 1.0 is also reproduced in Appendix B, "Service Schema Files."

NOTE The *Liberty ID-SIS Employee Profile Service Specification* can be found at `http://www.projectliberty.org/specs/liberty-idsis-ep-v1.0.pdf`.

Data Services Template 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 ID-WSF-DST. Additional information on these interfaces can be found in “Data Services Template API” on page 118 and in the Javadocs at */AccessManager_base*`/SUNwam/docs`.

Liberty Personal Profile Service

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

NOTE Registering a service with the Discovery Service is done by updating a resource offering for that service. For more information, see Chapter 6, “Discovery Service.”

The 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 the user. The following steps detail the system process for the Liberty Personal Profile Service.

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 SOAP Request handler to process the request.

   The PersonalProfile RequestHandler processes the request based on the request type (either query or modify) and the query expression. This might entail the authorization of a WSC using Access Manager Policy Service. It might also make use of 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 it back to the WSC.

   a. For a response to a query request, the Liberty Personal Profile Service builds a personal profile container (as defined by the specification). This is an XML blob 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 these values will be used while building the XML container. The Personal Profile Service then applies `xpath` queries on the XML blob and gives us the resultant XML data node.

   b. For a response to a modify request, it parses the Modifiable Select expression and updates the new data from the new data node in the request.

Liberty Personal Profile Service Attributes

The Liberty Personal Profile Service attributes are *global* attributes. The values of these attributes are carried across the Sun Java System Access Manager configuration and inherited by every organization.

NOTE For information on the types of attributes used in Access Manager, see the Service Management chapter of the *Sun Java System Access Manager 6 2005Q1 Developer's Guide* (`http://docs.sun.com/doc/817-7649`).

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

- ResourceID Mapper
- Authorizer
- Attribute Mapper

- Provider ID
- Name Scheme
- Namespace Prefix
- Supported Containers
- PPLDAP Attribute Map List
- Require Query PolicyEval
- Require Modify PolicyEval
- Extension Container Attributes
- Extension Attributes Namespace Prefix
- Is ServiceUpdate Enabled
- Service Instance Update Class
- Alternate Endpoint

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 ID.

Authorizer

Before processing a request, the Liberty Personal Profile Service will verify the authorization of the WSC making the request. There are two levels of authorization check that can be done:

1. Is the requesting entity authorized to access the requested resource profile information?
2. Is the requested resource published to the requestor?

Authorization occurs via 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: `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 Personal Profile service. For example, a rule can be defined as follows:

Code Example 5-1 Authorization Rules

```
/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 also defined in the Liberty Personal Profile Service:

- Require Query PolicyEval
- Require Modify PolicyEval

Attribute Mapper

This 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`.

Provider ID

The value of this attribute defines the unique identifier for this instance of the Liberty Personal Profile Service. The format is:

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. You can choose from `First Last`, or `First Middle Last`.

Namespace Prefix

The value of this attribute specifies the namespace prefix 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 and is useful to distinguish metadata from different XML schema namespaces.

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 Service.

NOTE The term *container* as described here is not related to the Access Manager identity-related object also named *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 Currently, Access Manager has not made piublic this functionality.

PPLDAP Attribute Map List

Each identity attribute defined by the Liberty Personal Profile Service has a one-to-one match to an Access Manager User service attribute. The value of this attribute is a list that specifies those mappings. For example, `JobTitle=sunIdentityServerPPEmploymentIdentityJobTitle` maps the Liberty `JobTitle` attribute to the Access Manager `sunIdentityServerPPEmploymentIdentityJobTitle` attribute. When adding new attributes to either side of this equation, ensure that any new *attribute mappings* are configured in this attribute.

NOTE Attribute mappings are defined as global attributes under the name `sunIdentityServerPPDSAttributeMapList` in the Liberty Personal Profile Service XML service file definition. This “PPLDAP Attribute Map List” attribute corresponds to that `sunIdentityServerPPDSAttributeMapList` global attribute.

In Code Example 5-2, the Liberty Personal Profile Service `informalName` attribute mapping to the User service attribute `uid` is added to the mappings already present in the `amLibertyPersonalProfile.xml`.

Code Example 5-2 Attribute Mappings as Defined in XML Service File

```
<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>
             </AttributeSchema>
```

Require Query PolicyEval

If selected, this option requires a policy evaluation to be performed for Liberty Personal Profile Service queries.

Require Modify PolicyEval

If selected, this option requires a policy evaluation to be performed for Liberty Personal Profile Service modifications.

Extension Container Attributes

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

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

Code Example 5-3 illsutrates an extension query expression for *creditcard*, an extension attribute.

Code Example 5-3 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"
```

Code Example 5-3 Extension Query for creditcard *(Continued)*

```
   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.” This prefix is prepended to the element and is useful 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.

Liberty Employee Profile Service

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

NOTE Before implementing this example, you must have two instances of Access Manager installed, running, and Liberty-enabled. Completing the steps in “sample1” on page 170 of Appendix A, “Included Samples” 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 110. The Readme.html in the sample directory provides detailed steps on how to deploy and configure this sample for use as a data service. More information can be found in Appendix A, “Included Samples.”

Data Services Template API

The ID-WSF-DST specifies a base layer that can be extended by any instance of a data service. It defines how to query and modify data stored in a data service, and provides some common attributes that might be used in 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 service returns information associated with their identity. The information may include the individual’s office location and phone number, as well as other data such as job title and department name. From the implementation point of view, all identity services must be built on top of the ID-WSF-DST which provides the data model and message interfaces.

NOTE Figure 5-1 on page 108 illustrates how Access Manager uses the ID-WSF-DST as the framework for identity data services. The Liberty Web Services layer is the framework for creating, discovering and consuming identity data services, including a SOAP-based transport binding that allows identity services to be discovered and invoked. Other Liberty Web Services include the Discovery Service, and Interaction Service.

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

- com.sun.identity.liberty.ws.dst

- com.sun.identity.liberty.ws.dst.service

com.sun.identity.liberty.ws.dst

Table 5-1 summarizes the Data Services Template client APIs included in the com.sun.identity.liberty.ws.dst package.

Table 5-1 Data Service Client APIs

Class Name	Description
DSTClient	Provides common functions for the Data Service Templates query and modify option.
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 for DST modify request.
DSTQuery	Represents a Data Services Template query request.
DSTQueryItem	The wrapper for one query item for Data service.
DSTQueryResponse	Represents a Data Services Template query response.
DSTUtils	Provides utility methods used by the DST layer.

For more detailed API reference information, including methods and their syntax and parameters, see the Javadocs in */AccessManager_base*/SUNWam/docs.

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

The com.sun.identity.liberty.ws.dst.services package provides a handler class that can be used by any generic identity data service built using the *Liberty Alliance ID-SIS 1.0 Specifications*.

NOTE The Data Services is built using the *Liberty ID-SIS Personal Profile Service Specification*, based on the *Liberty Alliance ID-SIS 1.0 Specifications*.

The DSTRequestHandler 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 reference information, see the Javadocs in */AccessManager_base*/SUNWam/docs.

NOTE Access Manager provides a sample which makes use of the DSTRequestHandler interface. The sis-ep sample illustrates how to implement the DSTRequestHandler and deploy a new identity data service instance. It is located in the */AccessManager_base*/SUNWam/samples/phase2/sis-ep directory. "sis-ep" on page 172 of the Appendix A, "Included Samples" further discusses this sample.

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 based on the ID-WSF-DST. Sections 2 and 3 in the Readme.html provided in the */AccessManager_base*/SUNWam/samples/phase2/sis-ep directory have detailed steps on how to deploy and configure data services. But, in order to use those instructions for a new data service, you need to write a new data service schema. This XSD file (as discussed in Appendix B, "Service Schema Files") defines the service's data and data structure. Once this new XSD file is written, it can be used in place of the lib-id-sis-ep.xsd in the sample instructions to deploy your new data service.

CAUTION Instructions on writing the XSD service file are beyond the scope of this documentation.

Chapter 6

Discovery Service

The Sun Java™ System Access Manager contains an implementation of the "Discovery Service Specification" from the Liberty Alliance Project. The Discovery Service instance 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 contains the following topics:

- Overview
- Discovery Service Architecture
- Discovery Service Process
- Discovery Service Attributes
- Discovery Entries and Resource Offerings
- Discovery Service Interfaces
- Discovery Service Sample

Overview

The initial step in accessing identity data is to determine where the information is located. (For example, 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 (part of the *Liberty Identity Web Services Framework*) 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 The *Discovery Service Specification* can be found on the Liberty Alliance Project Web site at `http://www.projectliberty.org/specs/liberty-idwsf-disco-svc-v1.1.pdf`.

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 unique resource identifier (URI) registered with the discovery service that points to a particular discovery resource.

NOTE A discoverable service is assigned a service type URI in the specification that defines it. This URI points to a Web Services Description Language (WSDL) file that describes the service's data, the operations that can be perfomed on it, and a protocol detailing how to send it. The discoverable service specification itself adds the available ways the data can be exchanged.

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.

TIP Because a provider hosting the Discovery Service may 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, by providing 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 though 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 roll. This scenario only supports entry lookups. For more information on discovery entries, see "Discovery Entries and Resource Offerings" on page 132.

XML Service Files

The Discovery Service is defined using the XML service file `amDisco.xml`. `amDisco.xml` 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 More information on XML service files can be found in the section on XML Service Files in the *Sun Java System Access Manager 6 2005Q1 Developer's Guide* (`http://docs.sun.com/doc/817-7649`).

A second XML file, `amDisco_add.xml` (found in */AccessManager_base*`/SUNWam/upgrade/services50_sunIdentityServerDiscoveryService/10_20/data`), is used for upgrading Identity Server 6.2 to Access Manager 6.3. It lists the changes to the `amDisco.xml` file since 6.2.

NOTE More information on upgading and migration can be found in the *Java Enterprise System 2005Q1 Upgrade and Migration Guide* located at (`http://docs.sun.com/doc/817-7645`).

Application Programming Interfaces

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

- `com.sun.identity.liberty.ws.disco`
- `com.sun.identity.liberty.ws.disco.plugins`
- `com.sun.identity.liberty.ws.interfaces`

Additional information on these interfaces can be found in "Discovery Service Interfaces" on page 142 and in the Javadocs.

com.sun.identity.liberty.ws.disco

The `com.sun.identity.liberty.ws.disco` package includes a client application programming interface (API) that provides interfaces to communicate with the Discovery Service.

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

The `com.sun.identity.liberty.ws.disco.plugins` package includes an interface that can be used to develop plugins.

com.sun.identity.liberty.ws.interfaces

The `com.sun.identity.liberty.ws.interfaces` package includes interfaces that can be used to implement functionality common to all Liberty-enabled identity services in Sun Java System Access Manager. Several implementations of these interfaces have been developed for the Discovery Service.

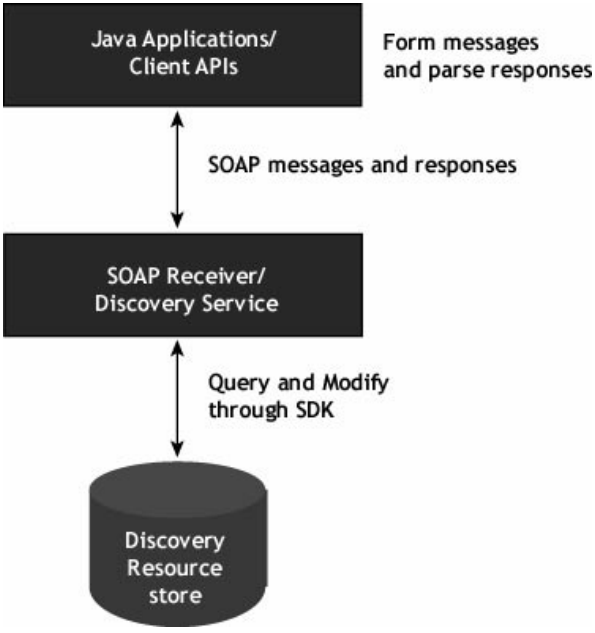
Discovery Service Architecture

The Access Manager Discovery Service includes Java and Web services-based interfaces. Java applications use the client API (discussed in "Client APIs" on page 145) 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 incorporating 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. Information on the SOAP Binding Service can be found in Chapter 7, "SOAP Binding Service."

The SOAP message is then sent on to the Discovery Service which parses a discovery resource identifier from it. This identifier is used to find the matching user DN which is then used to process the request. 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. Figure 6-1 on page 125 details this architecture.

Figure 6-1 Discovery Service Architecture

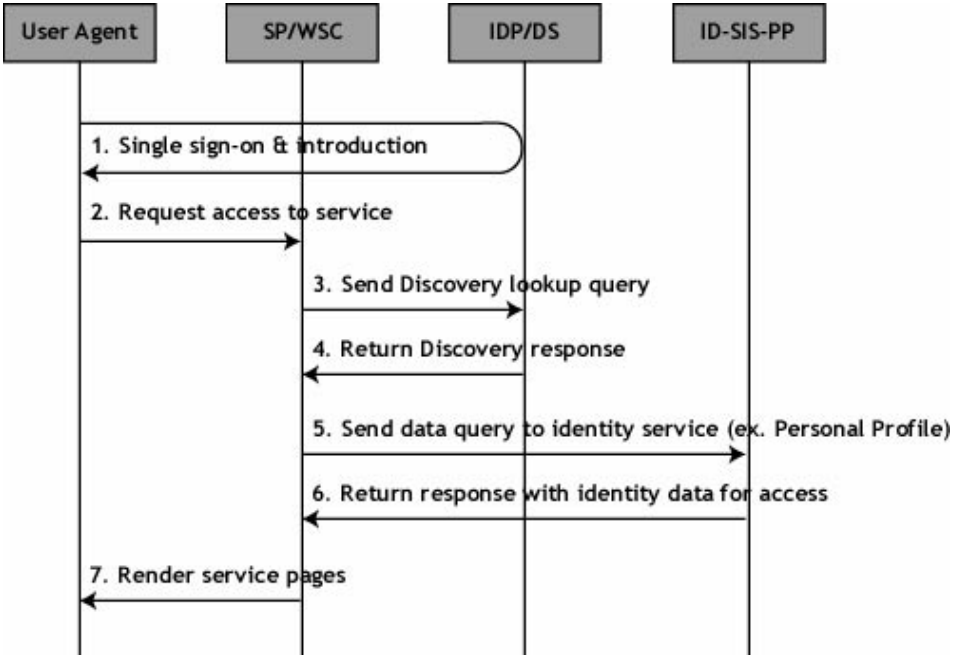


Discovery Service Process

Figure 6-2 provides a high-level overview of the interaction between parties in a Liberty-enabled Web services environment using the Discovery Service.

NOTE In Figure 6-2, the identity provider hosts the Discovery Service.

Figure 6-2 Liberty-enabled Discovery Service Process



The following steps detail the process illustrated in Figure 6-2.

1. The user logs onto 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:

    a. The user points their browser to 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 pass muster, the user is authenticated.

    d. Assuming the identity provider is the center of an authentication domain, it will notify authenticated principals that they have the option to federate any local identities created with member organizations. The principal would then accept or decline this invitation to federate. By accepting the invitation, the principal will be introduced to the option of federation everytime they log on to a member organization's Web site. If they accept this federation option, single sign-on is enabled.

2. After authentication, the user now requests access to services hosted by another service provider in the authentication domain.

   Single sign-on authentication in this step requires contacting the user's Personal Profile service via information from the Discovery Service.

3. The service provider sends a lookup query to the Discovery Service.

   Information used by any client to contact Discovery Service is culled from the authentication statement returned in Step 1.

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) 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.

   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. The Discovery Service attributes are:

- Provider ID

- Supported Authentication Mechanisms
- Supported Directives
- Enable Policy Evaluation for DiscoveryLookup
- Enable Policy Evaluation for DiscoveryUpdate
- Authorizer Plugin Class
- Entry Handler Plugin Class
- Classes For ResourceIDMapper Plugin
- Authenticate Response Message
- Generate SessionContextStatement for Bootstrapping
- Encrypt NameIdentifier in Session Context for Bootstrapping
- Use Implied Resource; don't generate ResourceID for Bootstrapping
- Resource Offerings for Bootstrapping Resources

Provider ID

This attribute takes as a value a URI that points to the Discovery Service. The value is written in the format:

`http://`*`host`*`:`*`port`*`/amserver/Liberty/disco`

Supported Authentication Mechanisms

This attribute specifies the authentication methods supported by the Discovery Service. 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. Table 6-1 details the available options.

Table 6-1 Policy-related Directives

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

CAUTION The `AuthorizeRequestor` and `EncryptResourceID` directives can not be used together.

Enable Policy Evaluation for DiscoveryLookup

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

Enable Policy Evaluation for DiscoveryUpdate

If selected, the service will perform a policy evaluation for the `DiscoveryUpdate` operation. By default, this option is not selected.

Authorizer Plugin Class

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

Entry Handler Plugin Class

The value of this attribute is the name and path to the plugin class that implements the `DiscoEntryHandler` interface used to set or retrieve a principal's discovery entries. A default implementation is provided for the Access Manager Discovery Service. 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.

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` (format: *providerID* + "/" + *the Base64 encoded userIDs*)
- `com.sun.identity.liberty.ws.disco.plugins.DefaultHexResourceIDMapper` (format: *providerID* + "/" + *the hex string of userID*)

Different implementations may be developed with the implementing class and added as a value of this attribute by clicking Add and using the following format:

`providerid=`*providerID*`|`*class_name_and_path*

The value of `providerid` is a key / value pair separated by a pipe (“|”).

Authenticate Response Message

If selected, the service will authenticate the response message. By default, the option is not selected.

Generate SessionContextStatement for Bootstrapping

If selected, the specifies whether to generate a `SessionContextStatement` for bootstrapping. `SessionConxtext` in the `SessionContextStatement` is needed by the Discovery Service to support the `AuthenicateSessionContext` directive. By default, this option is not selected.

Encrypt NameIdentifier in Session Context for Bootstrapping

If selected, the service will encrypt the name identifier in a `SessionContextStatement`. By default, the option is not selected.

Use Implied Resource; don't generate ResourceID for Bootstrapping

If selected, the service will not generate a resource identifier for bootstrapping. By default, the option is not selected.

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 regarding the Discovery Service. For more information defining on resource offerings, see “Discovery Entries and Resource Offerings.”

CAUTION The value of the Resource Offerings for Bootstrapping Resources attribute is a default value configured during installation of Access Manager. If you wish to define a new resource offering, click New. If you wish to edit an existing resource offering, click Edit.

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 which 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 an organization or a role. This scenario only supports entry lookups. More information can be found in:

- Storing Discovery Entries as User Attributes
- Storing Discovery Entries as Dynamic Attributes
- Storing Discovery Entries for Bootstrapping

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 details how to access and create a user's resource offerings.

1. Choose Users from the View menu in the Navigation pane of the Identity Management module.
2. Click on the Properties arrow next to the user for whom you wish to create (or modify) a resource offering.
3. Choose Resource Offering from the View menu in the Data pane.
4. Click New to access the resource offering attributes.
5. Enter a value for the Resource ID Attribute.

   This field defines an optional identifier for the resource offering.

6. Enter the Resource ID Value.

   This required field defines the resource identifier. *Resource identifiers* are URIs registered with the Discovery Service that point to a particular discovery resource. 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 in circumstances where there is only one resource that 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.

7. Enter a description of the resource offering in the Abstract field.

   This field is optional.

8. 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:id-sis-pp:2003-08`

9. 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://profile-provider.com`

10. Click New 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 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. Define a value for the Conrete Service Description attributes by selecting either the Brief SoapHttp Description radio button or the WSDL Reference radio button.

To configure Brief SoapHttp Description (selected by default):

I. Select Brief SoapHttp Description to provide the information necessary to invoke basic SOAP-over-HTTP-based service instances without using WSDL.

II. Enter a value for the SOAP-over-HTTP end point in the End Point attribute field.

This field contains the URI of the SOAP-over-HTTP endpoint. The URI scheme must be HTTP or HTTPS as in:

`https://soap.profile-provider.com/soap`

III. Enter a value for the SOAP action in the SOAP Action attribute field.

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

To configure WSDL Reference:

I. Select WSDL Reference to in order to reference a concrete WSDL service instance file.

II. Enter a value for the Required Field WSDL URI attribute.

This field contains the URI of the WSDL document.

III. Enter a value for the Required Field Service Namespace attribute.

This field references a `wsdl:service` element with the WSDL resource, such that `ServiceNameRef` is equal to the `wsdl:name` attribute of the proper `wsdl:service` element.

IV. Enter a value for the Service Local Part attribute.

This field provides the local portion of the qualified name of the service namespace URI.

NOTE WSDL Reference is not currently supported in the client.

11. Add a URI to specify any options for the resource offering.

This field lists the options available for the resource offering. Options provide hints to a potential requestor concerning the availability of certain data or operations to a particular offering. The set of possible URIs are defined by the service type and not the discovery service. If no option is specified, the service instance does not advertise any available options.

NOTE The *Liberty ID-SIS Personal Profile Service Specification* standardizes a set of options. This specification can be found at `http://www.projectliberty.org/specs/liberty-idsis-pp-v1.0.pdf`.

12. 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:

a. **GenerateBearerToken**. This directive specifies that a bearer token be generated.

b. **AuthenticateRequester**. This directive must be used with any service description that use SAML for message authentication.

c. **EncryptResourceID**. This directive specifies that the Discovery Service encrypt the resource ID.

d. **AuthenticateSessionContext.** This directive is specified when a Discovery Service provider includes a SAML assertion containing a `SessionContextStatement` in any future `QueryResponse` messages.

e. **AuthorizeRequester.** This directive is specified when a Discovery Service provider wants to include a SAML assertion containing a `ResourceAccessStatement` in any future `QueryResponse` messages.

NOTE If you wish to associate a directive with one or more service descriptions, select the checkbox in front of that Description ID. If no service descriptions are selected, the directive is applied to all description elements in the resource offering.

13. Click Save.

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 an organization. The role or organization can then be assigned to an identity-related object making the entry available to all users within the object. To create a discovery entry as a dynamic attribute, the Discovery Service must first be added and a template for the service created.

NOTE For more information on adding a service and creating a template, see the *Sun Java System Access Manager Administration Guide* (`http://docs.sun.com/doc/817-7647`).

After a service has been added and a template created, the procedure is the same as that detailed in "Storing Discovery Entries as User Attributes" on page 132 except for the following:

1. Select the Identity Management module in the Header frame.
2. Choose Roles from the View menu in the Navigation pane.
3. Click on the Properties arrow next to the role to which you want to add the discovery entry.
4. Choose Services from the View menu in the Data pane.
5. Click Edit next to the Discovery Service under the heading Service Configuration for this Role.
6. Select a priority level to resolve conflicting resource offerings.

   The conflict resolution level sets a priority level for roles that may contain the same user. For example, if User1 is assigned to both Role1 and Role2, you can define a higher priority level for Role1 so the resource offering from Role1 will be dominant.
7. Enter a description of the resource offering in the Abstract field.

   This field is optional.

8. 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:id-sis-pp:2003-08
   ```

9. 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://profile-provider.com
   ```

10. Click New 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 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. Define a value for the Conrete Service Description attributes by selecting either the Brief SoapHttp Description radio button or the WSDL Reference radio button.

       To configure Brief SoapHttp Description (selected by default):

       I. Select Brief SoapHttp Description to provide the information necessary to invoke basic SOAP-over-HTTP-based service instances without using WSDL.

       II. Enter a value for the SOAP-over-HTTP end point in the End Point attribute field.

           This field contains the URI of the SOAP-over-HTTP endpoint. The URI scheme must be HTTP or HTTPS as in:

           ```
           https://soap.profile-provider.com/soap
           ```

       III. Enter a value for the SOAP action in the SOAP Action attribute field.

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

To configure WSDL Reference:

I. Select WSDL Reference to in order to reference a concrete WSDL service instance file.

II. Enter a value for the Required Field WSDL URI attribute.

This field contains the URI of the WSDL document.

III. Enter a value for the Required Field Service Namespace attribute.

This field references a `wsdl:service` element with the WSDL resource, such that `ServiceNameRef` is equal to the `wsdl:name` attribute of the proper `wsdl:service` element.

IV. Enter a value for the Service Local Part attribute.

This field provides the local portion of the qualified name of the service namespace URI.

NOTE WSDL Reference is not currently supported in the client.

11. Add a URI to specify any options for the resource offering.

This field lists the options available for the resource offering. Options provide hints to a potential requestor concerning the availability of certain data or operations to a particular offering. The set of possible URIs are defined by the service type and not the discovery service. If no option is specified, the service instance does not advertise any available options.

NOTE The Liberty ID-SIS Personal Profile Service Specification standardizes a set of possible option values. This specification can be found at `http://www.projectliberty.org/specs/liberty-idsis-pp-v1.0.pdf`.

12. Select a directive for the resource offering.

You can choose from the following:

a. **GenerateBearerToken**. This directive specifies that a bearer token be generated.

b. **AuthenticateRequester**. This directive must be used with any service description that use SAML for message authentication.

c. **EncryptResourceID**. This directive specifies that the Discovery Service encrypt the resource ID.

d. **AuthenticateSessionContext.** This directive is specified when a Discovery Service provider includes a SAML assertion containing a `SessionContextStatement` in any future `QueryResponse` messages.

e. **AuthorizeRequester.** This directive is specified when a Discovery Service provider wants to include a SAML assertion containing a `ResourceAccessStatement` in any future `QueryResponse` messages.

13. Click Save.

CAUTION Unlike storing a discovery entry as a user attribute, this scenario only supports entry lookups, not updates.

Storing Discovery Entries for Bootstrapping

When a WSC contacts the Discovery Service for a resource offering, the WSC first needs to find the Discovery Service itself. 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 details how to configure a global attribute for bootstrapping the Discovery Service itself.

1. Select the Service Management module in the Header frame.
2. Click on the Properties arrow next to the Discovery Service in the Navigation pane.
3. Choose New under Resource Offerings for Bootstrapping Resources.

   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. Enter a description of the resource offering in the Abstract field.

   This field is optional.

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`

7. Click New 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 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. Define a value for the Conrete Service Description attributes by selecting either the Brief SoapHttp Description radio button or the WSDL Reference radio button.

      To configure Brief SoapHttp Description (selected by default):

      I. Select Brief SoapHttp Description to provide the information necessary to invoke basic SOAP-over-HTTP-based service instances without using WSDL.

      II. Enter a value for the SOAP-over-HTTP end point in the End Point attribute field.

         This field contains the URI of the SOAP-over-HTTP endpoint. The URI scheme must be HTTP or HTTPS as in:

         `https://soap.profile-provider.com/soap`

      III. Enter a value for the SOAP action in the SOAP Action attribute field.

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

To configure WSDL Reference:

I. Select WSDL Reference to in order to reference a concrete WSDL service instance file.

II. Enter a value for the Required Field WSDL URI attribute.

This field contains the URI of the WSDL document.

III. Enter a value for the Required Field Service Namespace attribute.

This field references a `wsdl:service` element with the WSDL resource, such that `ServiceNameRef` is equal to the `wsdl:name` attribute of the proper `wsdl:service` element.

IV. Enter a value for the Service Local Part attribute.

This field provides the local portion of the qualified name of the service namespace URI.

NOTE WSDL Reference is not currently supported in the client.

8. Add a URI to specify any options for the resource offering.

This field lists the options available for the resource offering. Options provide hints to a potential requestor concerning the availability of certain data or operations to a particular offering. The set of possible URIs are defined by the service type and not the discovery service. If no option is specified, the service instance does not advertise any available options.

NOTE The *Liberty ID-SIS Personal Profile Service Specification* standardizes a set of possible option values. This specification can be found at `http://www.projectliberty.org/specs/liberty-idsis-pp-v1.0.pdf`.

9. Select a directive for the resource offering.

You can choose from the following:

a. **GenerateBearerToken**. This directive specifies that a bearer token be generated.

b. **AuthenticateRequester**. This directive must be used with any service description that use SAML for message authentication.

c. **EncryptResourceID**. This directive specifies that the Discovery Service encrypt the resource ID.

d. **AuthenticateSessionContext.** This directive is specified when a Discovery Service provider includes a SAML assertion containing a `SessionContextStatement` in any future `QueryResponse` messages.

e. **AuthorizeRequester.** This directive is specified when a Discovery Service provider wants to include a SAML assertion containing a `ResourceAccessStatement` in any future `QueryResponse` messages.

10. Click Save.

Discovery Service Interfaces

By default, a discovery service is implemented as one of the identity web services in Access Manager. The Discovery Service provides the following implementations and interfaces:

- DefaultDiscoAuthorizer Implementation
- Default ResourceIDMapper Implementations
- DiscoEntryHandler Interface
- Client APIs

DefaultDiscoAuthorizer Implementation

The `com.sun.identity.liberty.ws.interfaces.Authorizer` is an interface used to enable an identity service to check the authorization of a WSC. The `DefaultDiscoAuthorizer` class is the default implementation of this interface. It 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. More information on this, and the Policy Service in general, can be found in the *Sun Java System Identity Server 2004Q2 Administration Guide* (`http://docs.sun.com/doc/817-7647`).

Policy definitions for the Discovery Service are configured using the Access Manager console. The procedure is as follows:

1. Choose Services from the View menu in the Navigation pane of the Identity Management module.

   The Discovery Service must be added to the organization for which the Discovery Service policy is being created. Proceed to Step 5 if this has already been done.

2. Click Add to add a new service to the organization.
3. Choose Discovery Service from the list of services in the Data Pane.
4. Click OK.
5. Choose Policies from the View menu in the Navigation pane of the Identity Management module.
6. Click New to create a new policy.
7. Select the type of policy.
8. Enter a name for the policy.
9. Click OK.
10. Choose Rules from the View menu in the Data pane for the created policy.
11. Click New.
12. Select Discovery Service for the rule type and click Next.
13. Enter a name for the rule.
14. Enter 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`

15. Select an action for the rule.

    Discovery Service policies can only look up or update data.

16. Click Finish.

NOTE The `com.sun.identity.liberty.ws.interfaces.Authorizer` interface can be implemented by any Web service in Access Manager. More information can be found in “Common Service Interfaces” on page 155 of Chapter 8, “Application Programming Interfaces” and in the Access Manager Javadocs (located in `/`*AccessManager_base*`/SUNWam/docs`.).

Default ResourceIDMapper Implementations

The `com.sun.identity.liberty.ws.interfaces.ResourceIDMapper` is an interface 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 ResourceID format to be:

  providerID + "/" + *the Base64 encoded userID*s

- `com.sun.identity.liberty.ws.disco.plugins.DefaultHexResourceIDMapper` assumes the ResourceID format to be:

  providerID + "/" + *the hex string of userID*.

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.

NOTE The `com.sun.identity.liberty.ws.interfaces.ResourceIDMapper` interface is common to all identity services in Access Manager not only the Discovery Service. More information can be found in “Common Service Interfaces” on page 155 of Chapter 8, “Application Programming Interfaces” and in the Access Manager Javadocs (located in `/`*AccessManager_base*`/SUNWam/docs`.).

DiscoEntryHandler Interface

The `com.sun.identity.liberty.ws.disco.plugins.DiscoEntryHandler` is an interface 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 in the Discovery Service. The default implementations of this interface are:

UserDiscoEntryHandler. This implementation 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 Personal Profile service.

DynamicDiscoEntryHandler. This implementation 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 Employee Profile service.

UserDynamicDiscoEntryHandler. This implementation 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

Table 6-2 summarizes the client APIs in the package `com.sun.identity.liberty.ws.disco`. For detailed API reference, including methods and their syntax and parameters, see the Javadocs in */AccessManager_base*`/SUNWam/docs`.

Table 6-2 Discovery Service Client APIs

Class Name	Description
`Description`	Represents a Description Type of a service instance.
`Directive`	Represents a discovery service DirectiveType element.
`DiscoveryClient`	Provides methods to send Discovery Service query and modify.
`EncryptedResourceID`	Represents an Encryption Resource ID element for the Discovery Service.
`InsertEntry`	Represents a Insert Entry for Discovery Modify request.
`Modify`	Represents a discovery modify request.
`ModifyResponse`	Represents a discovery response for modify request.
`Query`	Represents a discovery Query object.
`QueryResponse`	Represents a response for a discovery query request.

Table 6-2 Discovery Service Client APIs *(Continued)*

Class Name	Description
`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 via a service instance complying with one of the specified service type.
`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 outlining the process involved in querying and modifying the Discovery Service is included with Access Manager. It is located in the *AccessManager_base*`/SUNWam/samples/phase2/wsc` directory. The sample initally details how to deploy and run a WSC. The final portion queries the Discovery Service and modifies identity data in the Liberty Personal Profile Service. More information can be found in Appendix A, “Included Samples.”

Chapter 7

SOAP Binding Service

The Sun Java™ System Access Manager contains an implementation of the *Liberty ID-WSF SOAP Binding Specification* from the Liberty Alliance Project. SOAP Binding is a transport layer for sending and receiving SOAP messages. This chapter contains the following topics:

- Overview
- SOAP Binding Process
- SOAP Binding Attributes
- SOAP Binding Interfaces

Overview

The *Liberty Identity Web Services Framework* (ID-WSF) and *Liberty Identity Service Interface Specification* (ID-SIS) components of the Liberty Alliance Project (LAP) specifications use messages to convey identity data between providers. These *identity messages* themselves do not address a specific method of transport so Access Manager has implemented the *Liberty ID-WSF SOAP Binding Specification* (ID-WSF-SBS) 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 The Liberty ID-WSF *SOAP Binding Specification* can be found on the Liberty Alliance Project Web site at `http://www.projectliberty.org/specs/liberty-idwsf-soap-binding-v1.1.pdf`.

XML Service File

The Access Manager SOAP Binding service is defined using the XML service file `amSOAPBinding.xml`. `amSOAPBinding.xml` defines the attributes for the SOAP Binding service which can be managed through the Access Manager console or the XML file itself.

NOTE More information on XML service files can be found in the section on XML Service Files in the *Sun Java System Access Manager 6 2005Q1 Developer's Guide* (`http://docs.sun.com/doc/817-7649`).

The ID-WSF-SBS also defines an XML schema for use in building the SOAP messages. This XML Schema Defintion (XSD) file can be found on the LAP Web site. Version 1.0 is also reproduced in Appendix B, "Service Schema Files."

Application Programming Interfaces

The Access Manager SOAP Binding service includes a Java package named `com.sun.identity.liberty.ws.soapbinding`. Additional information on these interfaces can be found in "SOAP Binding Interfaces" on page 152.

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; in effect, a SOAP Receiver servlet.

NOTE Currently, only the Discovery Service, the Liberty Personal Profile Service and the Authentication Web Service use the SOAP Binding Service client API. Additionally, the Liberty Employee Profile sample uses them. They are not yet public.

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 to send this second message to the corresponding identity service for a response.

NOTE The Request Handler is an interface that must be implemented on the server side by any Liberty-based identity Web service using the SOAP Binding Service. More information on this interface can be found in the “Request Handler List” on page 150.

The identity 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 identity service for processing.

NOTE Before invoking a corresponding service, the SOAP framework might also do the following:

1. Authenticate sender identity: This is to verify the credentials of a WSC peer, probably by verifying it's client certificate.
2. Authenticate invoking identity: This verifies 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.
3. Granular authorization: This is to authorize the WSC itself before processing a service request.

SOAP Binding Attributes

The SOAP Binding service attributes are *global* attributes. The values of these attributes are carried across the Sun Java System Access Manager configuration and inherited by every organization.

NOTE For information on the types of attributes used in Access Manager, see the Service Management chapter of the *Sun Java System Access Manager Developer's Guide* (`http://docs.sun.com/doc/817-7649`).

Attributes for the SOAP Binding service are defined in the `amSOAPBinding.xml` service file. The SOAP Binding attributes are:

- Request Handler List
- Web Service Authenticator
- Supported Authentication Mechanisms

Request Handler List

The SOAP Binding Service provides the `RequestHandler` interface to process the request message and return a response. This interface must be implemented on the server side by each Liberty-based identity service that uses the SOAP Binding Service. The Request Handler List attribute stores information about the implementation classes of the Web services that implement the Request Handler.

NOTE Currently, only the Discovery Service, the Liberty Personal Profile Service and the Authentication Web Service use the SOAP Binding Service RequestHandler interface. Additionally, the Liberty Employee Profile Service sample uses it. The interface itself is not yet public.

The Request Handler List displays entries that contain key / value pairs separated by a pipe (“ | ”) as in:

```
key=disco|class=com.example.identity.liberty.ws.disco.DiscoveryService
```

key Parameter

The required *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 is:

protocol: / / *hostname: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: / / *hostname:port* / *amserver* / Liberty / disco

Different service clients use different keys when connecting to the SOAP Receiver.

class Parameter

The required *class* parameter specifies the name of the Request Handler implementation class for the particular identity service. For example:

```
class=com.example.identity.liberty.ws.disco.DiscoveryService
```

Web Service Authenticator

This attribute takes as a value the implementation class for the Web Service Authenicator interface. This class authenticates a request and generates a credential for a Web service consumer (WSC).

NOTE This interface is not currently 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 one is not selected, and a Web services consumer (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 More complete information on authentication mechanisms and their level of security can be found in the *Liberty ID-WSF Security Mechanisms* document on the Liberty Alliance Project Web site at `http://www.projectliberty.org/specs/liberty-idwsf-security-mechanisms-v1.1.pdf`.

SOAP Binding Interfaces

The Access Manager SOAP Binding Service includes a Java package named `com.sun.identity.liberty.ws.soapbinding`. It provides classes to construct SOAP requests and responses and to change the contact point for the SOAP binding. Table 7-1 details the available classes.

Table 7-1 SOAP Binding API Classes

Class	Description
`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.

The package also includes a `RequestHandler` interface. The implementation of this interface is discussed further in "Interaction Service API" on page 159 of Chapter 8, "Application Programming Interfaces." For more detailed API reference information, see the Javadocs in */AccessManager_base*`/SUNWam/docs`.

NOTE Be sure to check out Appendix A, "Included Samples" for sample code and files to help you understand the implementation of the Liberty Alliance Project's specifications in Access Manager.

Chapter 8

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 exposed graphical user interfaces for Liberty-based Web services (discussed in the Web services section of this book) as well as application programming interfaces (APIs). This chapter details information on the APIs that do not have a corresponding graphical user interface (GUI) and contains the following sections:

- Overview of Public Interfaces
- Common Service Interfaces
- Common Security API
- Interaction Service API
- PAOS Binding

Overview of Public Interfaces

Table 8-1 lists 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 with a GUI are described in the corresponding chapters of this book; links to those chapters are provided in the **Description** column. Packages that are used in the back-end are described in this chapter; links to those sections are also provided in the **Description** column. For detailed API reference, including methods and their syntax and parameters, see the Javadocs in */AccessManager_base*/SUNWam/docs.

Table 8-1 Summary of Liberty-based Packages

Package Name	Description
com.sun.identity.liberty.ws.authnsvc	Provides classes to manage the Authentication Web Service. See Chapter 4, “Authentication Web Service” on page 101.
com.sun.identity.liberty.ws.authnsvc.protocol	Provides classes to manage Authentication Web Service protocol. See Chapter 4, “Authentication Web Service” on page 101.
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 155 of this chapter.
com.sun.identity.liberty.ws.common.wsse	Provides an interface to parse and create a X.509 Certificate Token Profile. See “Interaction Service API” on page 159 of this chapter.
com.sun.identity.liberty.ws.disco	Provides interfaces to manage the Discovery Service. See Chapter 6, “Discovery Service” on page 121.
com.sun.identity.liberty.ws.disco.plugins	Provides a plugin interface for the Discovery Service. See Chapter 6, “Discovery Service” on page 121.
com.sun.identity.liberty.ws.dst	Provides classes to implement an identity service on top of the Access Manager framework. See Chapter 5, “Data Services” on page 107 for information on a service built using this API and for more general information.
com.sun.identity.liberty.ws.dst.service	Provides a handler class that can be used by any generic identity data service. See Chapter 5, “Data Services” on page 107 for information on data services and for more general information.
com.sun.identity.liberty.ws.interaction	Provides classes to support the Interaction RequestRedirect Profile. See “Interaction Service API” on page 159 of this chapter.
com.sun.identity.liberty.ws.interfaces	Provides interfaces common to all Access Manager Liberty-based Web service components.See Chapter 6, “Discovery Service” on page 121 and Chapter 5, “Data Services” on page 107 for information on default implementations. See “Common Service Interfaces” on page 155 of this chapter 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 161 of this chapter.
com.sun.identity.liberty.ws.security	Provides interface to manage Liberty-based Web service security mechanisms. See “Common Security API” on page 157 of this chapter.

Table 8-1 Summary of Liberty-based Packages *(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 7, “SOAP Binding Service” on page 147.
com.sun.liberty	Provides interfaces common to the Access Manager Federation Management module. See Chapter 3, “Federation Management” on page 59.

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 are:

- com.sun.identity.liberty.ws.common
- com.sun.identity.liberty.ws.interfaces

com.sun.identity.liberty.ws.common

The com.sun.identity.liberty.ws.common package includes classes common to all Liberty-based Access Manager service components.

Table 8-2 Common Liberty Classes

Class	Description
LogUtil	Class that defines methods which are used by the Liberty component of Access Manager to write logs.
Status	Class that represents a common status object.

For more detailed API reference information, see the Javadocs in */AccessManager_base*/SUNWam/docs.

com.sun.identity.liberty.ws.interfaces

The com.sun.identity.liberty.ws.interfaces package includes interfaces that can be implemented to add their corresponding functionality to each Liberty-based Access Manager Web service.

Table 8-3 Common Liberty Interfaces

Interface	Description
Authorizer	Interface for identity service to check authorization of a WSC.
ResourceIDMapper	Interface used to map between a userID and the ResourceID associated with it.

Authorizer

The com.sun.identity.liberty.ws.interfaces.Authorizer is an interface that, once implemented, can be used by each Liberty-based Web service component for access control.

NOTE The DefaultDiscoAuthorizer class is the implementation of this interface for the Discovery Service. For more information, see Chapter 6, "Discovery Service." The com.sun.identity.liberty.ws.idpp.plugin.IDPPAuthorizer class is the implementation for the Liberty Personal Profile Service. For more information, see Chapter 5, "Data Services."

The Authorizer interface enables the Web service to check for the authorization of a Web service consumer (WSC) to access the requested resource. When a WSC contacts a Web service provider (WSP), the WSC conveys a sender identity and an invocation identity. (The *invocation identity* is always the subject of the SAML assertion.) These conveyances allow 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 6 2005Q1 Developer's Guide* (http://docs.sun.com/doc/817-7649) for more information on policy management, single sign-on and sessions. See the *Sun Java System Access Manager 6 2005Q1 Administration Guide* (http://docs.sun.com/doc/817-7647) for information on creating policy.

ResourceIDMapper

The `com.sun.identity.liberty.ws.interfaces.ResourceIDMapper` is an interface used to map a user DN 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 ResourceID format to be:

  providerID + "/" + *the Base64 encoded userID*s

- `com.sun.identity.liberty.ws.disco.plugins.DefaultHexResourceIDMapper` assumes the ResourceID format to be:

  providerID + "/" + *the hex string of userID*.

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

The `com.sun.identity.liberty.ws.security` package includes an interface and classes to manage Liberty-based security mechanisms.

Table 8-4 com.sun.identity.liberty.ws.security

Class Name	Description
`SecurityTokenProvider`	A provider interface for managing Web Service Security (WSS) type tokens.
`ProxySubject`	Represents the identity of a proxy, the confirmation key and confirmation obligation the proxy must possess and demonstrate for authentication purpose
`ResourceAccessStatement`	Conveys information regarding the accessing entities and the resource for which access is being attempted

Table 8-4 com.sun.identity.liberty.ws.security *(Continued)*

Class Name	Description
SecurityAssertion	Provides an extension to the Assertion class to support ID-WSF ResourceAccessStatement and SessionContextStatement
SecurityTokenManager	This is the entry class for the security package com.sun.identity.liberty.ws.security. You can call its methods to generate X509 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	Class that defines methods which are used to get certificates and sign messages.
SessionContext	Represents session status of an entity to another system entity.
SessionContextStatement	An element that conveys session status of an entity to another system entity within the body of an <saml:assertion> element.
SessionSubject	Represents a liberty subject with associated session status.

For more detailed API reference information, see the Javadocs in */AccessManager_base*/SUNWam/docs.

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

The com.sun.identity.liberty.ws.common.wsse package includes APIs for creating security tokens used for authentication and authorization in accordance with the *Liberty ID-WSF Security Mechanisms* specification. This document can be found at the Liberty Alliance Project (LAP) Web site at http://www.projectliberty.org/specs/liberty-idwsf-security-mechanisms-v1.1.pdf . Both WSS X509 and SAML tokens are supported.

Table 8-5 Security APIs

Class Name	Description
BinarySecurityToken	The class BinarySecurityToken provides interface to parse and create X.509 Security Token depicted by Web Service Security: X.509
WSSEConstants	

For more detailed API reference information, see the Javadocs in */AccessManager_base*/SUNWam/docs.

Interaction Service API

It is often necessary for providers of identity services to interact with the owner of a reosource to get the resource owner's consent to expose data, or to get additional data from the resource owner. The Liberty Alliance Project (LAP) has defined the *Liberty ID-WSF Interaction Service Specification* to specify how these interactions can be carried out. Of the options for this interaction defined in the specification, Access Manager has implemented one of them: the *RedirectRequest*. In this profile, the Web service provider (WSP) requests the connecting Web service consumer (WSC) to redirect the user agent (principal) to an interaction resource (URL) at the WSP. Once the user agent sends an HTTP request to fetch the URL, the WSP has the opportunity to present one or more pages to the principal with questions for other information. When the WSP obtains the information it requires 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

There is no XML service file for the Interaction Service. There are two properties, though, that are configured upon installation in the `AMConfig.properties` file located in */AccessManager_base*/SUNWam/lib.

- `com.sun.liberty.ws.interaction.wspRedirectHandler` —This property points to the URL at which the WSPRedirectHandler servlet is deployed. The servlet handles the service provider side of interactions for user redirects.
- `com.sun.identity.liberty.interaction.wscSpecifiedInteractionChoice` —This property indicates the level of interaction in which the WSC will participate if they participate in user redirects. Possible values include `interactIfNeeded`, `doNotInteract`, and `doNotInteractForData`. The affirmative `interactIfNeeded` is the default.
- `com.sun.identity.liberty.interaction.wscWillIncludeUserInteractionHeader` —This property indicates whether the WSC will include a SOAP header to indicate certain preferences for interaction based on the LAP specifications. The default value is `yes`.

- com.sun.identity.liberty.interaction.wscWillRedirect —This property indicates whether the WSC will participate in user redirections. The default value is yes.
- com.sun.identity.liberty.interaction.wscSpecifiedMaxInteractionTime —This property indicates the maximum length of time (in seconds) the WSC is willing to wait for the WSP to complete their portion of the interaction. The WSP would not then initiate an interaction if the interaction is likely to take more than the time specified. For example, if the WSP receives a request where this property is set to a maximum 30 seconds and their own property com.sun.identity.liberty.interaction.wspRedirectTime (see below) is set to 40 seconds, the WSP will return a SOAP fault (timeNotSufficient) indicating that the time is not sufficient for interaction.
- com.sun.identity.liberty.interaction.wscWillEnforceHttpsCheck—This property indicates whether the WSC will enforce HTTPS in redirected URLs. The default value, yes, indicates that the WSC will not redirect the user when the value of redirectURL (specified by the WSP) is not an HTTPS URL.
- com.sun.identity.liberty.interaction.wspWillRedirect—The WSP can initiate 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—The WSP can initiate 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—This property indicates the length of time (in seconds) the WSP expects to take to complete an interaction and return control back to the WSC. For example, if 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 and wspRedirectTime is set to 40 seconds, the WSP will return a SOAP fault (timeNotSufficient) indicating that the time is not sufficient for interaction.
- com.sun.identity.liberty.interaction.wspWillEnforceHttpsCheck—This property indicates whether the WSP will enforce a HTTPS returnToURL specified by the WSC. The default value is yes.

- com.sun.identity.liberty.interaction.wspWillEnforceReturnToHostEqualsRequestHost—This property indicates whether the WSP would enforce the address values of returnToHost and requestHost if they are the same. Per the LAP specifications, the value of this property is always yes.
- com.sun.identity.liberty.interaction.htmlStyleSheetLocation—This property points to the location of the style sheet used to render the interaction page in HTML.
- com.sun.identity.liberty.interaction.wmlStyleSheetLocation—This property points to the location of the style sheet 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 these classes to interact with a resource owner. Table 8-6 details the API.

Table 8-6 Interaction Service API

Class	Description
InteractionManager	This class provides the interface and implementation for resource owner interaction.
InteractionUtils	This class provides some utility methods related to resoource owner interaction.
JAXBObjectFactory	This object contains factory methods. An ObjectFactory allows you to programatically construct new instances of the Java representation for XML content.

For more detailed API reference information, including methods and their syntax and parameters, see the Javadocs in */AccessManager_base*/SUNWam/docs.

PAOS Binding

Access Manager has implemented the optional Liberty Alliance Project (LAP) *Liberty Reverse HTTP Binding for SOAP Specification*. It defines a message exchange protocol that permits a 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 valuable 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.

PAOS vs. SOAP

In a typical SOAP binding, an HTTP client interacts with an identity service via a client request and a server response. For example, a cell phone user (client) may contact his phone service provider (service) in order 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 a HTTP response. The subsequent response from the client is bound to a request. This is why the reverse HTTP for SOAP binding is also known as PAOS; the spelling of SOAP is reversed.

PAOS Binding API

The Access Manager implementation of PAOS binding includes a Java package named `com.sun.identity.liberty.ws.paos`. It provides classes to parse a PAOS header, make a PAOS request, and receive a PAOS response.

NOTE These APIs are used by PAOS clients on the HTTP server side. APIs 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.

Table 8-7 details the available classes in `com.sun.identity.liberty.ws.paos`. For more detailed information, including methods and their syntax and parameters, see the Javadocs in */AccessManager_base*`/SUNWam/docs`.

Table 8-7 Summary of PAOS APIs

Class Name	Description
`PAOSHeader`	The PAOSHeader class is used by a web application on the HTTP server side to parse a PAOS header in an HTTP request from the user agent side.

Table 8-7 Summary of PAOS APIs

Class Name	Description
PAOSRequest	The PAOSRequest class is used by a web application on the HTTP server side to construct a PAOS request message and send it via an HTTPresponse to the user agent side.
PAOSResponse	The PAOSResponse class is used by a web application on the HTTP server side to receive and parse a PAOS response via an HTTP request from the user agent side.

Note that PAOSRequest is made available in PAOSResponse to provide correlation if needed by API users.

PAOS Binding Sample

A sample demonstrating PAOS service interaction between a 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 both included in the paos directory. Code Example 8-1 is the PAOS client servlet also included.

Code Example 8-1 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";
```

Code Example 8-1 PAOS Client Servlet from PAOS Sample *(Continued)*

```
    res.setContentType("text/plain");
    res.setContentLength(1+msg.length());
    PrintWriter out = new PrintWriter(res.getOutputStream());
    out.println(msg);
    out.close();

    throw new ServletException(pe1.getMessage());
      }

      HashMap servicesAndOptions = paosHeader.getServicesAndOptions();

      Set services = servicesAndOptions.keySet();

      String thisURL = req.getRequestURL().toString();
      String[] queryItems = { "/IDPP/Demographics/Birthday" };
      PAOSRequest paosReq = null;
      try {
    paosReq = new PAOSRequest(thisURL,
              (String)(services.iterator().next()),
              thisURL,
              queryItems);
      } catch (PAOSException pe2) {
    pe2.printStackTrace();
    throw new ServletException(pe2.getMessage());
      }
      System.out.println("PAOS request to User Agent side
--------------->");
      System.out.println(paosReq.toString());
      paosReq.send(res, true);
  }

  public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {

      PAOSResponse paosRes = null;
      try {
    paosRes = new PAOSResponse(req);
      } catch (PAOSException pe) {
    pe.printStackTrace();
    throw new ServletException(pe.getMessage());
      }

      System.out.println("PAOS response from User Agent side
-------------->");
      System.out.println(paosRes.toString());

      System.out.println("Data output after parsing -------------->");

      String dataStr = null;
      try {
    dataStr = paosRes.getPPResponseStr();
      } catch (PAOSException paose) {
    paose.printStackTrace();
    throw new ServletException(paose.getMessage());
```

Code Example 8-1 PAOS Client Servlet from PAOS Sample *(Continued)*

```
        }
        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();
    }
}
```

NOTE Be sure to check out Appendix A, “Included Samples” for information on all the sample code and files included with Access Manager.

Part III

Appendices

Appendix A

Included Samples

Sun Java™ System Access Manager has included a number of samples that make use of the Liberty Alliance Project's specifications and its own implementations of said documents. This appendix contains information regarding the Liberty-based samples. It includes the following sections:

- Overview
- Federation Framework Samples
- Web Services Framework Samples

Overview

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.

Federation Framework Samples

Access Manager 2005Q1 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 sub-directories:

- sample1
- sample2
- sample3

sample1

The `sample1` sample provides a collection of files, located in the */AccessManager_base*`/SUNWam/samples/liberty/sample1` directory, 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) and an identity provider (IDP). Each needs to be deployed and configured on different Access Manager installations. Table A-1 contains relative information for the two required servers.

Table A-1 Relative 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
		Personal Profile Service

The `Readme.html` in the sample directory provides detailed steps on how to deploy and configure this sample. In addition, the procedures and additional information are written up in “Federation Management Samples” on page 88 of Chapter 3, “Federation Management.”

NOTE Sample1 also contains instructions for configuring a common domain. For information on common domains, see “Common Domain” on page 31 of Chapter 1, “Introduction to the Liberty Alliance Project” and “Common Domain Services” on page 65 of Chapter 3, “Federation Management.”

sample2

The `sample2` sample provides a collection of files, located in the */AccessManager_base*`/SUNWam/samples/liberty/sample2` directory, to configure a basic environment for creating and managing a federation but, in this case, the resources of the SP are deployed on a Sun Java System Web Server protected by an Access Manager Policy Agent. As in sample1, the SP and IDP are deployed and configured on different Access Manager installations. Apart from highlighting account federation, SSO, single logout, and federation termination, this sample also

demonstrates 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` in the sample directory provides detailed steps on how to deploy and configure this sample.

sample3

The `sample3` sample provides a collection of files, located in the */AccessManager_base*`/SUNWam/samples/liberty/sample3` directory, 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 the Access Manager. Because of this, 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 the 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.

Since this scenario involves multiple IPs, you will also need to install a Common Domain Service. This service can be installed on the same machine as the Access Manager software or on a different machine. The `Readme.html` in the sample directory provides detailed steps on how to deploy and configure this sample. In addition, information on common domains can be found in “Common Domain Services” on page 65 of Chapter 3, “Federation Management.”

Web Services Framework Samples

Access Manager 6 2005Q1 supports both the *Liberty Alliance Identity Web Services Framework 1.0 Specifications* and the *Liberty Alliance Identity Services Interface Specifications 1.0*. These 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
- sis-ep

- paos
- authnsvc

wsc

The `wsc` sample provides a collection of files, located in the `/`*AccessManager_base*`/SUNWam/samples/phase2/wsc` directory, to deploy and run a Web service consumer (WSC).

NOTE Before implementing this example, you must have two instances of Access Manager installed, running, and Liberty-enabled. Completing the steps in “sample1” on page 170 will accomplish this.

In addition, this sample illustrates how to use the Discovery Service and Data Service Template client APIs to allow the WSC to communicate with a Web service provider (WSP). (The WSP is the Liberty Personal Profile Service installed with Access Manager.) It details the flow of the Liberty-based Web Service Framework (ID-WSF), and how the security mechanisms and interaction service come into play. The `Readme.html` in the sample directory provides detailed steps on how to deploy and configure this sample. In addition, information can be found in Chapter 6, “Discovery Service” and Chapter 5, “Data Services.”

sis-ep

The `sis-ep` sample provides a collection of files, located in the `/`*AccessManager_base*`/SUNWam/samples/phase2/sis-ep` directory, to develop, deploy and invoke a new Liberty-based Web service to Access Manager. The sample implements a Liberty-based Employee Profile Service.

NOTE Before implementing this example, you must have two instances of Access Manager installed, running, and Liberty-enabled. Completing the steps in “sample1” on page 170 will accomplish this.

The Employee Profile Service is a deployment of the *Liberty ID-SIS Employee Profile Service Specification* (ID-SIS-EP) which is itself an instance of the *Liberty Alliance ID-SIS 1.0 Specifications*. The `Readme.html` in the sample directory provides detailed steps on how to deploy and configure this sample. In addition, related information can be found in Chapter 5, “Data Services.”

paos

The paos sample provides a collection of files, located in the */AccessManager_base*/SUNWam/samples/phase2/paos directory, to demonstrate how to set up and invoke a PAOS Service interaction between a client and server. (In a real-world deployment, the server-side code would be developed by a service provider.) The sample is based on the following scenario: a cell phone user subscribes to a news service offered by his cell phone's manufacturer. The news service automatically pushes 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.

NOTE You can also see the output from PAOSClientServlet program in the log file of the Web Server. For example, when using Sun Java System Web Server, look in the log subdirectory for the errors file.

The Readme.html in the sample directory provides detailed steps on how to deploy and configure this sample. In addition, information can be found in "PAOS Binding Sample" on page 163 of Chapter 8, "Application Programming Interfaces."

authnsvc

The authnsvc sample provides a collection of files, located in the */AccessManager_base*/SUNWam/samples/phase2/authnsvc directory, 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 in the sample directory provides detailed steps on how to deploy and configure this sample. In addition, information can be found in "Authentication Web Service Sample" on page 105 of Chapter 4, "Authentication Web Service."

Appendix B

Service Schema Files

This appendix contains some of the XML Schema Definition (XSD) files discussed in this document. It includes the following sections:

- Overview
- SOAP Binding Schema
- Personal Profile Schema
- Employee Profile Schema
- Authentication Web Service Schema
- PAOS Binding Schema
- Metadata Description Schema

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 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 may be stored in implementation-specific ways, but will be exposed

by the service using the XML schema (specified here), and the Web Services Description Language definition of the service type (not specified in this documentation set). The XSD files in this appendix are reproduced here for your convenience. They (and a number of other XSD files) are also available on the Project Liberty Web site at http://www.projectliberty.org/resources/specifications.php.

SOAP Binding Schema

Code Example B-1 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 7, "SOAP Binding Service."

Code 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
```

Code Example B-1 SOAP Binding XSD File *(Continued)*

```
            Copyright (c) 2004 Liberty Alliance participants, see
            http://www.projectliberty.org/specs/idwsf_copyrights.html
        </xs:documentation>
    </xs:annotation>

    <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"/>
```

Code Example B-1 SOAP Binding XSD File *(Continued)*

```
</xs:schema>
```

Personal Profile Schema

Code Example B-2 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 5, "Data Services."

Code 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>
```

Code Example B-2 Personal Profile Service XSD File *(Continued)*

```
  <xs:simpleType name="SelectType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:element name="PP" type="PPType"/>
  <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"/>
      <xs:element ref="MN" minOccurs="0"/>
      <xs:element ref="LMN" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="Extension" minOccurs="0"/>
    </xs:sequence>
```

Code Example B-2 Personal Profile Service XSD File *(Continued)*

```
    <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>
      <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"/>
```

Code Example B-2 Personal Profile Service XSD File *(Continued)*

```
      <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"/>
      <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"/>
```

Code Example B-2 Personal Profile Service XSD File *(Continued)*

```
<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>
  </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"/>
```

Code Example B-2 Personal Profile Service XSD File *(Continued)*

```
    </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>
```

Employee Profile Schema

Code Example B-3 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 5, “Data Services.”

Code 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>
```

Code Example B-3 Employee Profile Service XSD Schema *(Continued)*

```
  <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"/>
      <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"/>
```

Code Example B-3 Employee Profile Service XSD Schema *(Continued)*

```
      <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"/>
  <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

Code Example B-4 is a reproduction of the `liberty-idwsf-authn-svc-v1.0.xsd`, the XSD file that accompanies Liberty ID-WSF Authentication Service Specification as discussed in Chapter 4, “Authentication Web Service.”

Code Example B-4 Authentication Web Service XSD File

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

<xs:schema
targetNamespace="urn:liberty:sa:2004-04"
```

Code Example B-4 Authentication Web Service XSD File *(Continued)*

```
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:sa="urn:liberty:sa:2004-04"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
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>
                      <xs:simpleContent>
                          <xs:extension base="xs:base64Binary"/>
                      </xs:simpleContent>
                  </xs:complexType>
              </xs:element>
```

Code Example B-4 Authentication Web Service XSD File *(Continued)*

```
            <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>

            <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>
```

Code Example B-4 Authentication Web Service XSD File *(Continued)*

```
            </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>

                        <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"/>
```

Code Example B-4 Authentication Web Service XSD File *(Continued)*

```
                </xs:complexType>
              </xs:element>
          </xs:sequence>
      </xs:complexType>
  </xs:element>

</xs:schema>
```

PAOS Binding Schema

Code Example B-5 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 structure of PAOS requests and responses. PAOS Binding is discussed in Chapter 8, "Application Programming Interfaces."

Code 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"
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"/>
```

Code Example B-5 Reverse HTTP Binding for SOAP XSD File *(Continued)*

```
    <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

Code Example B-6 is a reproduction of `liberty-metadata-1.0-errata-v2.0.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.

Code 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-errata-v2.0
```

Code Example B-6 Metadata Description and Discovery XSD File *(Continued)*

```
4 June 2004

Copyright (c) 2004 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"/>
      <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"/>
```

Code Example B-6 Metadata Description and Discovery XSD File *(Continued)*

```
      </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 name="libertyPrincipalIdentifier" type="entityIDType"
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">
      <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"/>
```

Code Example B-6 Metadata Description and Discovery XSD File *(Continued)*

```
      <xs:element name="RegisterNameIdentifierServiceReturnURL"
        type="xs:anyURI" minOccurs="0"/>
      <xs:element name="NameIdentifierMappingProtocolProfile"
        type="xs:anyURI" 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 name="validUntil" type="xs:dateTime"/>
    <xs:attribute name="cacheDuration" type="xs:duration"/>
  </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>
    <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"/>
```

Code Example B-6 Metadata Description and Discovery XSD File *(Continued)*

```
      <xs:element ref="Extension" minOccurs="0"/>
      <xs:element ref="ds:Signature" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="providerID" type="entityIDType" use="required"/>
    <xs:attribute name="id" type="xs:ID" use="optional"/>
    <xs:attribute name="validUntil" type="xs:dateTime"/>
    <xs:attribute name="cacheDuration" type="xs:duration"/>
  </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">
    <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"/>
```

Code Example B-6 Metadata Description and Discovery XSD File *(Continued)*

```
      <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 name="validUntil" type="xs:dateTime"/>
    <xs:attribute name="cacheDuration" type="xs:duration"/>
    <xs:attribute name="id" type="xs:ID" use="optional"/>
  </xs:complexType>
</xs:schema>
```

Glossary

For a list of terms used in this documentation set, refer to the latest *Sun Java™ Enterprise System Glossary*:

`http://docs.sun.com/doc/816-6873`

Index

E

F

I

J

L

M

N

O

P

R

S

T

U

W

X