Sun Java™ System

Identity Server Federation Management Guide

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

Part No: 817-6362-10

Contents

About This Guide

The *Sun Java™ System Identity Server Federation Management Guide* provides information about the Federated Management module of Identity Server 2004Q2 (formerly Sun™ ONE Identity Server). It includes an introduction to federation management and Identity Server's compliance to Liberty Alliance specifications. Instructions for enabling a Liberty II environment, and summaries of the APIs you can use to extend the Federation Management framework are also provided in this guide.

This preface includes the following topics:

Audience for This Guide

This *Federation Management Guide* is intended for use by IT administrators and software developers who implement an integrated 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™ technology
- JavaServer Pages™ (JSP) technology

- HyperText Transfer Protocol (HTTP)
- HyperText Markup Language (HTML)
- eXtensible Markup Language (XML)

Because Sun Java System Directory Server is used as the data store in an Identity Server deployment, administrators should also be familiar with the documentation provided with that product. The latest Directory Server documentation can be accessed online.

Identity Server 2004Q2 Documentation Set

The Identity Server documentation includes two sets:

- Identity Server Core Documentation
- Identity Server Policy Agent Documentation

Identity Server Core Documentation

The Identity Server documentation set contains the following titles:

- *Technical Overview* (http://docs.sun.com/doc/817-5706) provides a high-level overview of how Identity Server components work together to consolidate identity management and to protect enterprise assets and web-based applications. It also explains basic Identity Server concepts and terminology.
- *Migration Guide* (http://docs.sun.com/doc/817-5708) provides details on how to migrate existing data and Sun Java System product deployments to the latest version of Identity Server. (For instructions about installing Identity Server and other products, see the *Sun Java Enterprise System 2004Q2 Installation Guide* (http://docs.sun.com/doc/817-5760).
- *Administration Guide* (http://docs.sun.com/doc/817-5709) describes how to use the Identity Server console as well as manage user and service data via the command line.
- *Deployment Planning Guide* (http://docs.sun.com/doc/817-5707) provides information on planning an Identity Server deployment within an existing information technology infrastructure.

- *Developer's Guide* (http://docs.sun.com/doc/817-5710) offers information on how to customize Identity Server 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/817-5711) provides summaries of data types, structures, and functions that make up the public Identity Server C APIs.
- *Federation Management Guide* (http://docs.sun.com/doc/817-6362) provides information about Federation Management, which is based on the Liberty Alliance Project.
- The *Release Notes* (http://docs.sun.com/doc/817-5712) 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.

Updates to the *Release Notes* and links to modifications of the core documentation can be found on the Identity Server page at the Sun Java System documentation web site (http://docs.sun.com/prod/entsys.04q2). Updated documents will be marked with a revision date.

Identity Server Policy Agent Documentation

Policy agents for Identity Server documents are available on this Web site:

http://docs.sun.com/coll/S1_IdServPolicyAgent_21

Policy agents for Identity Server are available on a different schedule than the server product itself. Therefore, the documentation set for the policy agents is available outside the core set of Identity Server documentation. The following titles are included in the set:

- *Web Policy Agents Guide* documents how to install and configure an Identity Server 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 Identity Server policy agent that can 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.

Updates to the *Release Notes* and modifications to the policy agent documentation can be found on the Policy Agents page at the Sun Java System documentation web site. Updated documents will be marked with a revision date.

Your Feedback on the Documentation

Sun Microsystems and the Identity Server technical writers are interested in improving this documentation and welcomes your comments and suggestions. Use the following web-based form to provide feedback to us:

`http://www.sun.com/hwdocs/feedback/`

Please provide the full document title and part number in the appropriate fields. The part number can be found on the title page of the book or at the top of the document, and is usually a seven or nine digit number. For example, the part number of the Federation Management Guide is 817-6362.

Documentation Conventions Used in This Guide

In the Identity Server documentation, certain typographic conventions and terminology are used. These conventions are described in the following sections.

Typographic Conventions

This book uses the following typographic conventions:

- *Italic type* is used within text for book titles, new terminology, emphasis, and words used in the literal sense.
- `Monospace font` is used for sample code and code listings, API and language elements (such as function names and class names), filenames, pathnames, directory names, HTML tags, and any text that must be typed on the screen.

- *Italic serif font* is used within code and code fragments to indicate variable placeholders. For example, the following command uses *filename* as a variable placeholder for an argument to the `gunzip` command:

  `gunzip -d` *filename*`.tar.gz`

Terminology

The following terms are used in the Identity Server documentation set:

- *Identity Server* refers to Identity Server and any installed instances of the Identity Server software.
- *Policy and Management services* refers to the collective set of Identity Server components and software that are installed and running on a dedicated deployment container such as a web server.
- *Directory Server* refers to an installed instance of Sun Java System Directory Server.
- *Application Server* refers to an installed instance of Sun Java System Application Server (also known as Sun ONE Application Server.)
- *Web Server* refers to an installed instance of Sun Java System Web Server (also known as Sun ONE Web Server).
- *Web container that runs Identity Server* refers to the dedicated J2EE container (such as Web Server or Application Server) where the Policy and Management Services are installed.
- *IdentityServer_base* represents the base installation directory for Identity Server. The Identity Server 2004Q2 default base installation and product directory depends on your specific platform:
  - Solaris™ systems: `/opt/SUNWam`
  - Linux systems: `/opt/sun/identity`

  The product directory is `/SUNWam` for Solaris systems and `/identity` for Linux systems. When you install Identity Server 2004Q2, you can specify a different directory for `/opt` on Solaris systems or `/opt/sun` on Linux systems; however, do not change the `/SUNWam` or `/identity` product directory.

  For the base installation directory of the following products, refer to the documentation for the specific product.

- *DirectoryServer_base* represents the base installation directory for Sun Java System Directory Server.
- *ApplicationServer_base* is a variable place holder for the home directory for Sun Java System Application Server.
- *WebServer_base* is a variable place holder for the home directory for Sun Java System Web Server.

Related Information

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
- Download Center:
  http://wwws.sun.com/software/download/
- Technical Support:
  http://www.sun.com/service/sunone/software/index.html
- Professional Services:
  http://www.sun.com/service/sunps/sunone/index.html
- Sun Enterprise Services, Solaris Patches, and Support:

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

Related Third-Party Web Site References

Third-party URLs are referenced in this document 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.

Chapter 1

Introduction to Identity Federation and the Web Services Framework

This chapter explains the concept of identity federation, and describes the role of the Federation Management module in Sun™ Java System Identity Server 2004Q2.

The Need for Federated Identities

Consider the many times an individual accesses services on the Internet in a single day. At work, he uses the company intranet to perform a multitude of business-related tasks such as reading and sending email, looking up information in the company phone book and other internal databases, and submitting expense reports and other business-related online forms. At home after work, he checks his personal email, then logs into an online news service to check his baseball team's standings. He may finalize his travel plans via his travel agent's website, and then does some online shopping at his favorite clothing store. Each time he accesses a service on the Internet, he must log in and identify himself to the service provider.

A *local identity* refers to the set of attributes or information that identify a user to a particular service provider. These attributes typically include a name and password, plus an email address, account number or other identifier. For example, the individual in our scenario is known to his company's network as an employee number, but he is known to his travel agent as Joe Smith. He is known to his online

news service by an account number, and he is known to his favorite clothing store by a different account number. He uses one email name and address for his personal email, and a different email name and address for his workplace. Each of these different user names represents a different local identity.

Identity federation allows a user to consolidate the many local identities he has configured among multiple service providers. With one *federated identity*, the individual can log in at one service provider's site and move to an affiliated service provider site without having to re-authenticate or re-establish his identity. For example, with a federated identity, the individual might want to access both his personal email account and his business email account from his workplace, and move back and forth between the two services without having to log in each time. Or at home he might want to log in to an online clothing store, then access the online news service, and communicate with his travel agent. It is a convenience for the user to be able to access all of these services without having to provide different user names and passwords at each service site. It is a valuable benefit to the user when he can do so safely, and knowing that his identity information is secure.

The Liberty Alliance Project was implemented to make this possible.

The Liberty Alliance Project

In 2001 Sun Microsystems joined with other major companies to form the Liberty Alliance Project, the premier open standards organization for federated identity and identity-based services. The Liberty Alliance Project develops specifications and guidelines for implementing complete network identity infrastructures and for deploying identity-based web services.

The members of the Liberty Alliance Project represent some of the world's most recognized brand names and service providers, driving products, services and partnerships across a spectrum of consumer and industrial products, financial services, travel, retailing, telecommunications and technology. For more information including listings of Liberty web service products, specifications, cased studies, and white papers, see the Liberty Alliance Project website:

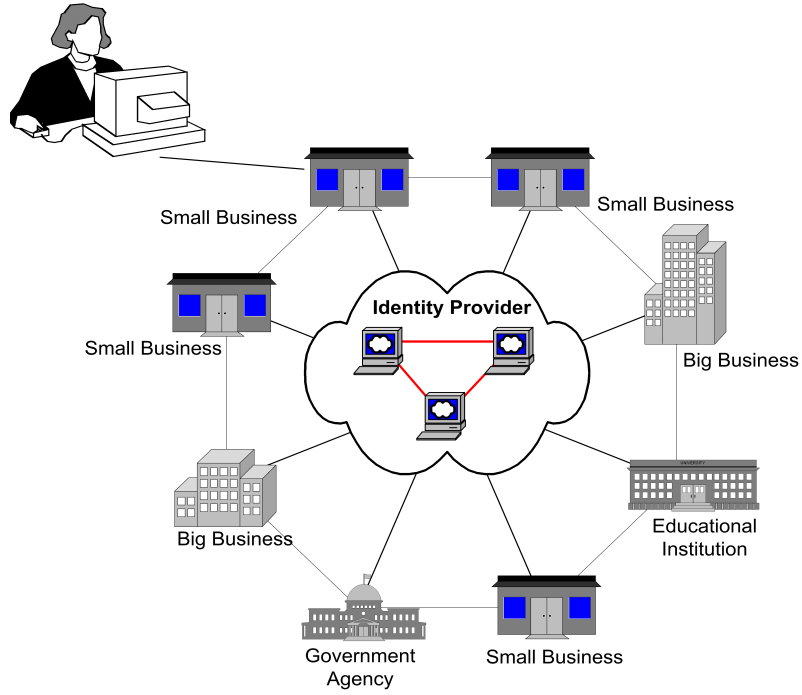
http://projectliberty.org/

The Circle of Trust

The goal of the Liberty Alliance Project is to enable individuals and organizations to easily conduct network transactions while protecting the individual's identity. This goal can be achieved only when commercial and non-commercial organizations join together into a *circle of trust*. In a circle of trust, service providers agree to join together in order to exchange user authentication information using Liberty web service technologies. This circle of trust must contain at least one identity provider, a service that maintains and manages identity information. The circle of trust also includes service providers that offer web-based services to users; it also includes users themselves. Once a Circle Of Trust is established, single sign-on is enabled between all the providers.

The circle of trust is also known as an *authentication domain*, although it is not a DNS domain or a domain in the Internet sense of the word. Figure 1-1 illustrates a user accessing a small business, a service provider that is associated with other businesses, and agencies in a circle of trust.

Figure 1-1 The circle of trust



Account Federation occurs when a user chooses to unite distinct service accounts and identity provider accounts. The user retains individual account information with each provider in the circle. At the same time, the user establishes a link that allows the exchange of authentication information between them. Users can choose to federate any or all identities they might have with the service providers that have joined this circle. When a user successfully authenticates with one service provider, she can access any of the her accounts within the circle of trust in a single session *without having to reauthenticate.*

Federation Management Architecture

The Identity Server 6.1 release implemented an Identity Federation Framework (ID-FF) version 1.1 Identity Server 2004Q2 adds new federation features defined in Identity Federation Framework (ID-FF) 1.2 specifications, including a web service framework to facilitate deployment of customer Identity Web Services. Client APIs are provided for web service consumers to communicate with web service providers.

The following three major components make it possible for identity information to be exchanged among service providers:

- Identity Federation Framework
- Identity Web Services Framework
- Identity Service Instance Specifications

Identity Federation Framework

The Identity Service Instance Specifications build upon the Web Service Framework and Federation Framework to provide services. The federation framework specifies core protocols, schema and concrete profiles that allow developers to create a standardized, multiple-vender, identity federation network. These include the following:

Opt-in account linking. Users can choose to federate different service provider accounts.

Authentication context. Service providers with federated accounts communicate the type and level of authentication that should be used when the user logs in.

Account linking termination. Users can choose to stop their account federation.

Identity provider introduction. This feature provides the means for service providers to discover which identity providers a *principal* uses. A principal can be an organization or individual who interacts with the system. This is important when there are multiple identity providers in an identity federation network.

Name Registration. This feature enables a service provider or identity provider to register with each other a new name identifier for a principal at any time following federation.

Single Sign-on and Federation Protocol A protocol that defines the process that a user at a service provider goes through to authenticate their identity with an identity provider. It also specifies the means by which a service provider obtains an Authentication Assertion from an identity provider to allow single sign-on to the user. There are two types of Single Sign-On which either the identity or service provider can implement:

- SOAP-based Single Sign On and Federation Protocol, which relies on a SOAP call from provider to provider. This is primarily the Browser Artifact SSO profile.
- Form POST-based Single Sign On and Federation Protocol, which rely on an HTTP form POST to communicate between providers.

Single Log-Out Protocol. TA protocol used to synchronize the session log-out functionality across all sessions that were authenticated and created by a particular identity provider. There are two types which either the identity or service provider can implement:

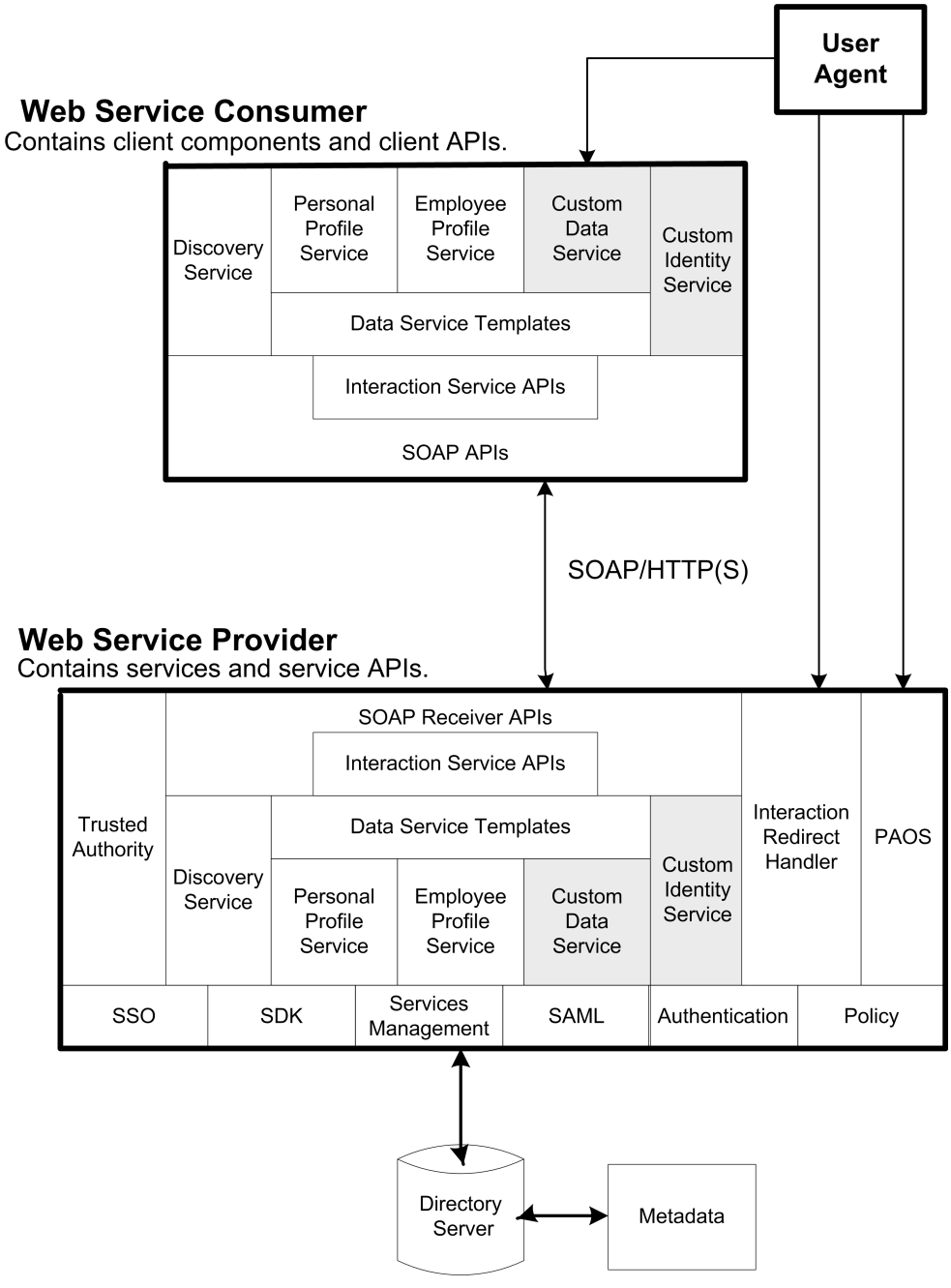
- SOAP-based Single Log-Out Protocol relies on asynchronous SOAP messaging calls between providers.
- HTTP Redirect-based Single Log-Out Protocol

Identity Web Services Framework

The Web Services Framework consists of a set of schema, protocols and profiles for providing a basic identity services, such as identity service discovery and invocation.

Three parties are required for identity federation in a basic Liberty Web Services environment: a user agent, a web service consumer, and a web service provider. Figure 1-2 illustrates the internal architectures of a Liberty Web Services Consumer and a Web Service Provider.

Figure 1-2 Liberty II Architecture.



The Web Service Consumer components and the Web Service Provider components are newly implemented components in Identity Server 2004Q2. The components in the bottom layer of the Web Service Provider were implemented in Identity Server 6.1.These components include Single-Sign On (SS0), the Identity Server SDK, Service Management Services, SAML, Authentication modules, and a Policy Service. In Figure 1-2, the Data Service and Identity Service represent custom services that you can add to the Web Services Framework.

The Web Services Framework consists of a set of schema, protocols and profiles for providing a basic identity services, such as identity service discovery and invocation. This framework is new in Liberty II and includes the following:

Discovery Service. An identity service that allows a requester to discover resource offerings. For more details, see “Discovery Service” on page 89.

SOAP Binding. A set of Java APIs for sending and receiving ID-* messages using SOAP and XML. “SOAP Binding” on page 86.

Data Services Template. A common data service layer for developing identity services. Includes common utilities for message error-checking and verification, and remote client APIs to support customer data service instances. For more details, see “Data Services Template” on page 91.

Security Mechanisms. Defines a set of authentication mechanism and security properties which are factored into authorization decisions enforced by the targeting identity-based web services. Each mechanism contains both peer entity authentication (null / TLS / CClientTLS) and message authentication (null / X509 / SAML). For more details, see “Running the Web Service Client” on page 71.

Interaction Service. A protocol for simple interaction of Web Services Framework participants with a Principal. For more details, see “Interaction Service” on page 97.

Trusted Authority. APIs for creating security tokens used for authentication and authorization in Liberty II-enabled services. For more details, see “Trusted Authority” on page 85.

Identity Service Instance Specifications

The Liberty Service Instance Specifications build upon the Web Services Framework and Federation Framework to provide services such as contacts, presence detection or wallet services that depend on network identity. The following Service Instance Specifications implementations are new in Identity Server 2004Q2.

Personal Profile Service. An instance of a data-oriented web service which offers profile information regrading a principal's personal life. A shopping portal that offers information such the principal's account number and shopping preferences is an example of a personal profile service. See "Personal Profile Service" on page 92.

Employee Profile Service. Similar to the Personal Profile Service, except that An instance of a data-oriented web service which offers profile information regarding a principal'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. See "Personal Profile Service" on page 92.

Supporting Components

Metadata Service. A library of command-line tools for loading metadata into the Identity Server data store. For more details, see "Metadata Specifications" on page 98.

Reverse HTTP Bindings. A protocol and set of APIs for retrieving data from Identity Server via clients such as cell phones. For more details, see "PAOS" on page 98.

The Federation Management Process

The Federation Management process begins with authentication. By default, Identity Server comes with two options for user authentication. The first is the proprietary Authentication Service; the second is the Liberty-enabled Federation Management module. With the first option, when a user tries to access a resource protected by Identity Server, he is redirected to the Authentication Service using an Identity Server login page. When the user provides credentials, the Authentication Service verifies his identity, and either allows or denies access.

With Liberty-enabled Federation Management, when a user attempts to access a resource protected by Identity Server, the authentication process begins with a search for a valid Identity Server session token. If a session token is found, the user is granted access to the requested page. The requested page, which belongs to a member of the circle of trust, contains a link which provides the user an opportunity to federate his identity. When the user clicks this link, he is directed through the Federation Single Sign-On Process process. If no session token is found, the user is directed through the Pre-Login Process.

Federation Single Sign-On Process

When a user signs on to access a protected resource or service, Identity Server sends a request to the identity provider for authentication confirmation. If the identity provider sends a positive response, the user gains access to all provider sites within the circle of trust. If the identity provider sends a negative response, the user is directed to authenticate again using the Federation Single Sign-On process.

In federated single sign-on, the user selects an identity provider and then sends his or her credentials for authentication. Once authentication is complete and access is granted, the user is redirected to the Identity Server Authentication Service. The user is automatically granted a session token and redirected to the requested page which contains a link to allow the user to federate his or her identity. As long as the session token remains valid, the user can access other services offered by other service providers in the circle of trust without having to sign on again.

Pre-Login Process

The purpose of the Pre-Login process is to establish a valid user session at the service provider site. When no Identity Server session token is found, the pre-login process begins with the search for another type of cookie, a Federation cookie.

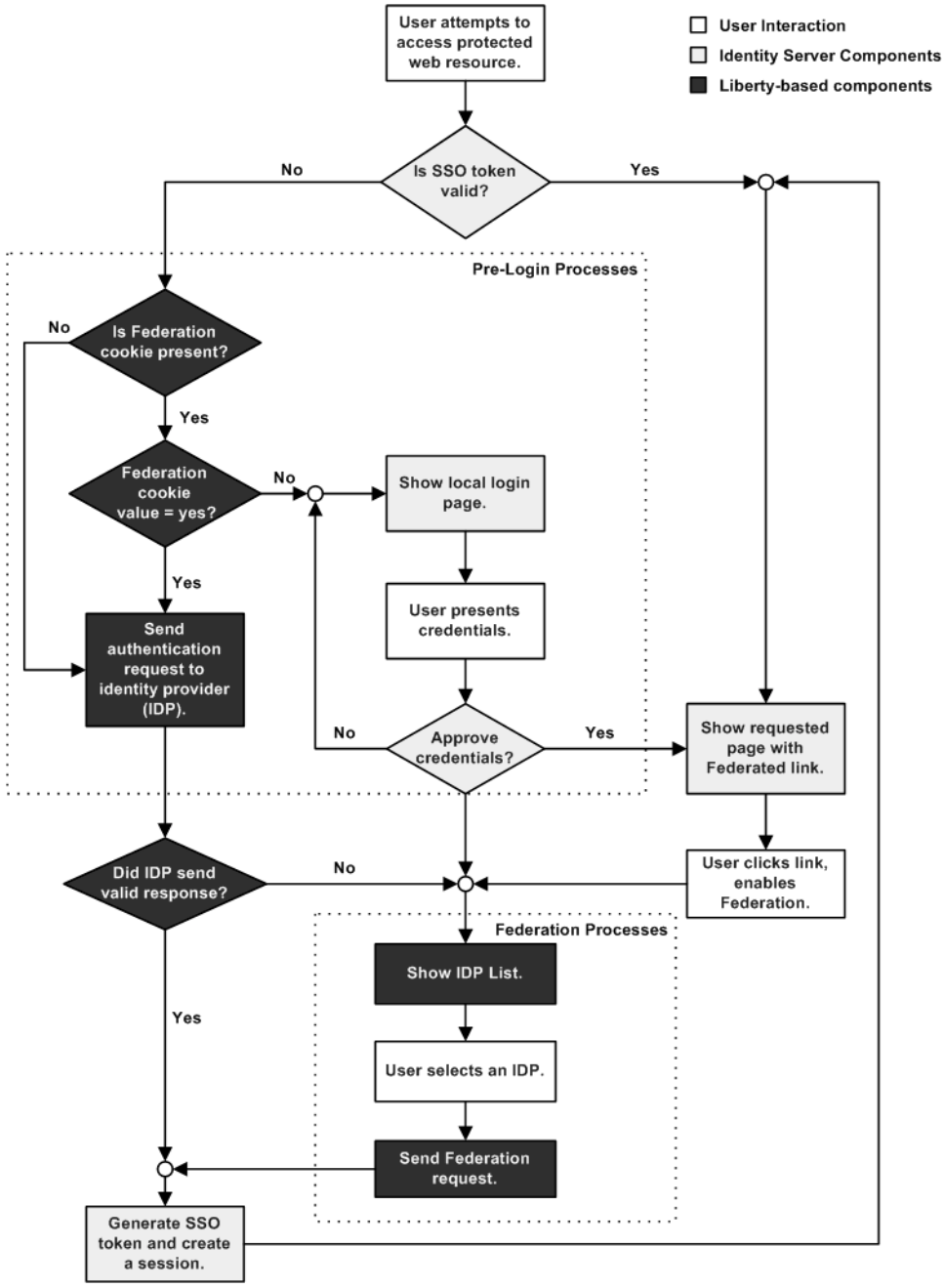
If, after the search for an Identity Server session token proves null, a Federation cookie is found and its value is “no,” an Identity Server login page is displayed and the user submits credentials to the Authentication Service. When authenticated by the Identity Server, the user is redirected to the requested page which contains a link to allow the user to federate their identity. If the user clicks this link, he is directed through the Federation Single Sign-On Process.

If, after the search for an Identity Server session token proves null, a valid Federation Cookie is found an its value is “yes,” it means the user has already been federated but not authenticated by an identity provider within the Circle of Trust. This is confirmed by sending a request for authentication to the user’s chosen identity provider.

If no Federation Cookie is found at all, a passive authentication request is sent to the user’s chosen identity provider. A passive authentication request does not allow identity provider interaction with the user. When an affirmative response is received back from the identity provider, the user is redirected to the Identity Server Authentication Service. There, the user is granted a session token and redirected to the requested page. When the response from the identity provider is negative (for example, if the session has timed out), the user is sent to a common login page where he can choose to do a local login or Federation Single Sign-On.

Figure 1-3 illustrates the differences between the Pre-Login process path and the Identity Federation path.

Figure 1-3 Liberty-enabled Identity Server Authentication Process Flow

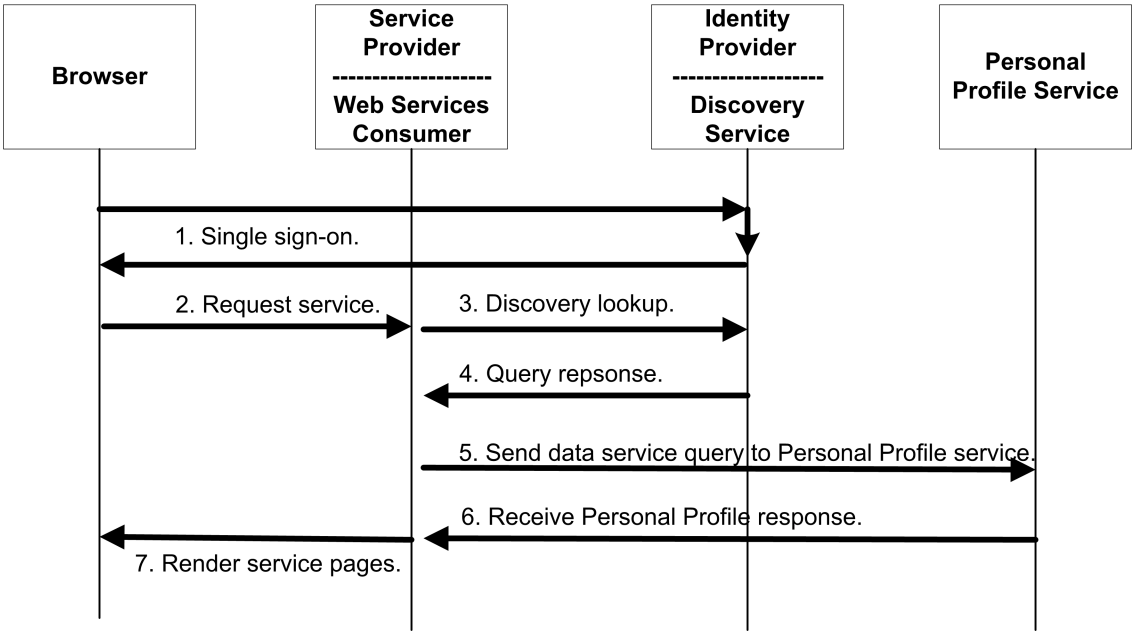


System Flow

Figure 1-4 provides a high-level view of the system flow between various parties in a Liberty web services environment. In this figure, note the following:

- The browser represents a *user agent*, a device used by an enterprise user.
- The Service Provider also acts as a Web Service Consumer.
- The Identity Provider hosts the Discovery Server.
- The Personal Profile Service represents a Web Service Provider.

Figure 1-4 The interaction between Liberty II components.



This is what happens on the back end when an employee looks up a colleague's phone number in an online corporate phone book:

1. The user's browser, Service Provider and Identity Provider complete the Federation Single-Sign-On process.

   An assertion with an attribute statement containing the Discovery Service *resource offering* will be included in the `ID-FF AuthnResponse`. This is the information used by any client to contact Discovery Service.

2. The user's browser requests access to services hosted on the Web Service Consumer.

   This requires contacting user's Personal Profile service.

3. The Web Service Consumer sends a discovery lookup query to the Discovery Service.

   The Web Service Consumer determines user's discovery resource offering from the bootstrap Assertion obtained in Step 1, then sends a discovery lookup query to the Discovery Service to determine where the user's Personal Profile instance is hosted.

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

   The lookup response contains the resource offering for the user's Personal Profile Service instance.

5. The Web Service Consumer sends a Data Services Template query to the SOAP end point of the Personal Profile Service instance.

   The query asks for the user's personal profile attributes, such as home phone number. The required authentication mechanism specified in the Personal Profile Service resource offering must be followed.

6. The Personal Profile Service instance authenticates and validates authorization or policy, or both, for the requested user or Web Service Consumer, or for both.

   If user interaction is required for some attributes, the Interaction Service will be invoked to query the user for consents or for attribute values. The Personal Profile Service instance returns a Data Services Template response to the Web Service Consumer after collecting all required data.

7. The Web Service Consumer processes the Personal Profile Service response, and then renders service pages containing the colleague’s contact information to the user's browser.

Chapter 2

Creating a Liberty Web Services Environment

Identity Server provides a collection of sample files, named Sample1, to illustrate how to configure basic identity federation using one Service Provider and one Identity Provider. The example demonstrates the basic use of various Liberty protocols including Account Federation, Single Sign-On, Single Logout, and Federation Termination.

Instructions for implementing Sample1 are described in the following sections and should be completed in this sequence:

- “Installing Identity Server” on page 30
- “Deploying the Service Provider” on page 30
- “Deploying the Identity Provider” on page 33
- “(Optional) Configuring a Third Level Domain” on page 36

You will find all Service Provider and Identity Provider files required to implement the examples in the following locations:

Table 2-1 Sample1 Directories

Directory Name Variable	Location
sample1_dir	*begin_dir*/samples/liberty/sample1/
sp1_sample_dir	*begin_dir*/samples/liberty/sample1/sp1/
idp1_sample_dir	*begin_dir*/samples/liberty/sample1/idp1/

Installing Identity Server

The first step in creating a Liberty environment is installing Identity Server on two separate machines. One Identity Server installation will act as a Service Provider, and one Identity Server installation will act as an Identity Provider. Follow the Identity Server installation instructions in the *Java Enterprise System Installation Guide*. Table 2-2 summarizes the variable placeholders and default values used in Sample 1.

Table 2-2 Default values in `metadata.xml` files for Sample1.

Installation Parameter	Service Provider Value	Identity Provider Value
Machine name	*machine1*	*machine2*
Solaris Installation Directory	*IS_Root*/SUNWam	*IS_Root*/SUNWam
Windows Installation Directory	*IS_Root*/SunONEIS	*IS_Root*/SunONEIS
Hostname variable	*SP1*	*IDP1*
Hostname	www.sp1.com	www.idp1.com
Identity Server Port	SERVER_PORT	SERVER_PORT
Identity Server Deployment URI	amserver	amserver
Identity Server 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
Metadata filename	sp1Metadata.xml	idp1Metadata.xml

Deploying the Service Provider

Before you can deploy the Service Provider, Identity Server must be installed and running over the HTTP(S) protocol. See "Installing Identity Server" on page 30.

Instructions for deploying the Service Provider are described in the following sections and should be completed in this sequence:

1. To Upload the Metadata for the Service Provider
2. To Configure the Service Provider
3. To Deploy the Service Provider.WAR File

To Upload the Metadata for the Service Provider

1. Update the following file with values appropriate to your Identity Server installation: *begin_dir*/samples/liberty/sample1/sp1/sp1Metadata.xml

   Default values are listed in Table 2-2 on page 30.

2. Load sp1Metadata.xml using following command:

   begin_dir/bin/amadmin -u amadmin -w password -t sp1Metadata.xml

To Configure the Service Provider

1. Locate the AMClient.properties file located in this directory:

   begin_dir/samples/liberty/sample1/sp1/WEB-INF/classes/

   Replace the following tags with values appropriate for your environment:

   SERVER_PROTO: Enter HTTPS or HTTP.

   SERVER_HOST: Enter the fully-qualified hostname for your Identity Server installation. Example: www.sp1.com

   SERVER_PORT: Enter the port number on which Identity Server is running.

   SERVICE_DEPLOY_URI: Enter the Identity Service 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.

2. Create the .war file for SP1.

   cd *sp1_sample_dir*
   jar -cvf sp1.war

3. Deploy the sp1.war file.

   See the next section.

To Deploy the Service Provider.WAR File

Choose one of the following as appropriate for your environment:

- If Identity Server is installed on Sun Java System Web Server
- If Identity Server is installed on Sun Java System Application Server

If Identity Server is installed on Sun Java System Web Server

1. Before you deploy a web application manually, be sure that the *server_root*/bin/https/httpsadmin/bin directory is in your path and that the IWS_SERVER_HOME environment variable is set to your server_root directory.
2. Enter the following command
   wdeploy deploy -u *uri_path* -i *instance* -v *vs_id* [-d *directory*] war_file
   - *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, or from which the application is deleted. If not specified for deployment, the application is deployed to the document root directory.
   - *war_file* is the WAR file name.

   Example:

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

3. Restart Web Server.

If Identity Server is installed on Sun Java System Application Server

1. Use the `asadmin deploy` command to deploy the WAR module. The syntax is as follows:

```
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] filepath
```

   For example, in Sample1 the `asadmin deploy` command deploys a web application as an individual module:

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

2. Restart Application Server.

Deploying the Identity Provider

Before you can deploy the Service Provider, Identity Server must be installed and running over the HTTP protocol. See “Installing Identity Server” on page 30.

The instructions for implementing Sampe1 are described in the following sections and should be followed in this sequence:

1. To Upload the Metadata for the Identity Provider
2. To Configure the Identity Provider
3. To Upload the Metadata for the Identity Provider

To Upload the Metadata for the Identity Provider

1. Update the following file as per your Identity Server installation:
   begin_dir/samples/liberty/sample1/idp1/sp1Metadata.xml

   Default values are listed in Table 2-2 on page 30.

2. Load idp1Metadata.xml using following command.

   begin_dir/bin/amadmin -u amadmin -w password -t idp1Metadata.xml

To Configure the Identity Provider

1. Locate the AMClient.properties file located in this directory:

   begin_dir/samples/liberty/sample1/idp1/WEB-INF/classes/

   Replace the following tags with values appropriate for your environment:

   SERVER_PROTO: Enter HTTP or HTTPS

   SERVER_HOST: Enter a fully-qualified hostname for your Identity Server installation. Example: www.idp1.com

   SERVER_PORT: Enter the port number where Identity Server is running.

   SERVICE_DEPLOY_URI: Enter the Identity Server 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.

2. Create the WAR file for IDP1.

   cd *idp1_sample_dir*
   jar -cvf idp1.war .

3. Deploy idp1.war.

   See the next section, "To Deploy the Service Provider.WAR File."

To Deploy the Identity Provider .WAR File

Follow the steps for one of the following:

- If Identity Server is installed on Sun Java System Web Server
- If Identity Server is installed on Sun Java System Application Server

If Identity Server is installed on Sun Java System Web Server

1. Before you can deploy a web application manually, you must make sure that the `server_root/bin/http/httpsadmin/bin` directory is in your path and that the `IWS_SERVER_HOME` environment variable is set to your `server_root` directory.
2. Enter the following command:

   ```
   wdeploy deploy -u uri_path -i instance -v vs_id [-d directory] war_file
   ```

   - *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, or from which the application is deleted. If not specified for deployment, the application is deployed to the document root directory.
   - *war_file* is the WAR file name.

   Example:

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

3. Restart the Web Server.

If Identity Server is installed on Sun Java System Application Server

1. Use the `asadmin deploy` command to deploy the WAR module. The syntax is as follows:

```
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] filepath
```

For example, the following command deploys a web application as an individual module:

```
asadmin deploy --user admin --password pswd1234 --host www.sp1.com
    --port 4848 --type web --contextroot IDP1 --instance server1 idp1.war
```

2. Restart Application Server.

(Optional) Configuring a Third Level Domain

The Sample1 application does not require the use of a third level domain. But if you want to configure third level domain, you can use the following instructions.

To Configure a Third-Level Domain

1. Access the Identity Server administration console.
2. Click the Federation tab.
3. Select Authentication Domain in the View menu, and then click Show.
4. Select **`sample1Alliance`** in the left pane.
5. In the right pane, enter the Reader Service URL and Writer Service URL as appropriate for your common domain services installation.

   For example, if common domain services are installed on machine3 with hostname is `www.machine3.com`, then for the default installation:

   Writer Service URL: `http://www.machine3.com:80/amcommon/writer`

   Reader Service URL: `http://www.machine3.com:80/amcommon/transfer`

Verifying a Successful Liberty Setup

The following sections provide detailed steps for using basic Liberty protocols to verify that you've successfully implemented Sample1:

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

SP1 has a protected page named `index.jsp`, and IDP1 has a protected page named `index.jsp`. Both protected pages include `_head.jsp`. The `_head.jsp` will check for a valid user session. If a session is invalid, the request is redirected to the Pre-Login service. The Pre-Login service attempts to perform a single sign-on (SSO). On first-time access, SSO will fail and the Pre-Login service redirects the request to the common Login page.

The protected page `index.jsp` contains the following three links:

Federate: Initiates the federation process.

Logout: Initiates the single logout process.

Terminate Federation: Initiates the federation termination process.

To Federate Service Provider and Identity Provider Accounts

1. Access the following URL in a web browser:

   SERVER_PROTO`//`*SERVER_HOST*`:`*SERVER_PORT*`/sp1/index.jsp`

   Example:

   `http://www.sp1.com:58080/sp1/index.jsp`

2. In the common Login page, click the Local Login link.

   You are redirected to the SP1's login page.

3. Log in to SP1.

   After successful authentication at SP1, the `index.jsp` is displayed.

4. Click the Federate link.

   The Federate page is displayed.

5. Select the preferred Identity Provider you want to federate with.

   In Sample1, select `IDP1` as your preferred Identity Provider. 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. This indicates that you have successfully federated your account between SP1 and IDP1.

Note that if the account is already federated, you will be redirected to the IDP login page.

To Perform a Single Sign-On

1. After successful federation (see previous section, "To Federate Service Provider and Identity Provider Accounts"), start a new browser session and access the SP1 protected page:

   SERVER_PROTO`://`*SERVER_HOST*`:`*SERVER_PORT*`/sp1/index.jsp`

   Example:

   `http://www.sp1.com:58080/sp1/index.jsp`

   The user is redirected to the IDP1 Login page.

2. 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

In either the SP1 protected page or the IDP protected page, `index.jsp`, click the Logout link.You are logged out from both SP1 and IDP1, and the LogoutDone page is displayed.

To Terminate Account Federation

1. In 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.

Deploying a Web Service Consumer

Deploying a web service consumer entails the following:

- Configuring the Service Provider
- Configuring the Identity Provider

- Running the Web Service Consumer Sample
- Interacting with the Personal Profile Service
- X.509 Message Authentication

Sample code for a web service consumer is provided with Identity Server to illustrate how to deploy and run a Liberty service client on top of the framework provided by Identity Server. All the files you need to deploy and initiate the service are located in the following directory:

install-dir/SUNWam/samples/phase2/wsc

NOTE Before you can implement this web service consumer example, you must have two Identity Servers already installed, running, and Liberty-enabled. Complete the steps in "Creating a Liberty Web Services Environment" on page 29 before you continue with this web service consumer example.

The Web Service Consumer Example

This example explains how you use the Discovery Service and Data Service Template client APIs to implement a web service client to communicate with a web service provider. In this example, the web service provider is the Personal Profile service that comes with Identity Server 2004Q2. The example demonstrates the flow of the Liberty Web Service Framework, and how the security mechanism and interaction service come into play in the federation process.

Once you've deployed and configured both a Service Provider and an Identity Provider, the Personal Profile service resides in the Identity Provider. Client code in the form of five `.jsp` files reside in the Service Provider. All the files you need to deploy and initiate the service are located in the following directory:

install-dir/SUNWam/samples/phase2/wsc

Table 2-3 summarizes the `.jsp` files provided with this example.

Table 2-3 JSPs provided in this sample

Name	Description
`index.jsp`	Retrieves boot strapping resource offering for discovery service.
`discovery-modify.jsp`	Adds Resource Offering for a user.

Table 2-3 JSPs provided in this sample

Name	Description
`discovery-query.jsp`	Sends query to discovery service for service resource offering.
`id-sis-pp-modify.jsp`	Sends `Data Service Modify` request to modify user attributes.
`id-sis-pp-query.jsp`	Sends `Data Service Query Request` to retrieve user attributes

Two machines are required for this sample:

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

There are five parties involved in this sample:

- Liberty Service Provider (SP)
- Liberty Identity Provider (IDP)
- Web Service Consumer (WSC)
- Liberty Discovery Service (DS)
- Liberty ID-SIS Personal Profile Service (ID-SIS-PP)

Configuring the Service Provider

1. Deploy the Liberty Sample1 Service Provider.

   Follow the instructions in the section "Deploying the Service Provider" on page 30.

2. Change the protocol support for the remote Identity Provider to ID-FF 1.2.

   a. Login to Identity Server Administration Console as Top-Level administrator.

   b. Click the Federation Management tab, and then in the View menu, choose Entity Descriptors.

   c. In the Entity ID list, choose the remote Identity Provider entity ID.

   d. In the right pane, in the View menu, choose Provider.l

   e. Under Provider, click the Edit link.

   f. Under the Protocol Support Enum field, choose `urn:liberty:iff:2003-08`.

   g. Click Save.

3. Replace the following tags and hostnames in `discovery-modify.jsp` and `index.jsp` with values appropriate for your environment.

   a. **IDP_SERVER_PORT:** Enter server port of the Identity Provider host.

   b. **SERVICE_DEPLOY_URI:** Enter the service deployment URI of the Identity Provider host.

   c. **`www.sp1.com`:** Enter the host name of the Service Provider machine if it is different from **`www.sp1.com`**.

   d. **`www.idp1.com`:** Enter host name of Identity Provider machine if different from **`www.idp1.com`**.

4. Deploy the Service Provider `.jsp` files.

   Copy all the five `.jsp` files to a subdirectory in the document root of the web container. For example, if the web container is, Sun Java System Web Server 6.1 run following command:

   ```
   mkdir webserber_install_root/docs/wsc
   cp is_install_root/SUNWam/samples/phase2/wsc/*.jsp
         webserber_install_root/docs/wsc/
   ```

5. Log in to the Identity Server.

6. Create a user named `spUser`.

   This user will be used as a federated user on the Service Provider side.

Configuring the Identity Provider

1. Deploy the Liberty Sample1 Identity Provider.

   Follow the instructions in the section "Deploying the Identity Provider" on page 33.

2. Register the Liberty Personal Profile Service.

   a. Log in to Identity Server as Top-Level administrator.

   b. In the Identity Management page, in the View menu, choose Service.

   c. Click Add.

   d. In the right pane, in the Available Services menu, choose Liberty Personal Profile Service.

   e. Click OK.

3. Create a user named `idpUser with` the Liberty Personal Profile Service enabled.

   This user will be used as the federated user on the Identity Provider side. It is also used for storing the Discovery Service resource offering and Personal Profile Service attributes.

Running the Web Service Consumer Sample

The following is an overview of what happens when you run the Web Service Client sample that comes with the product.

1. Complete the Liberty Single-Sign-On Process and obtain Discovery Service Boot Strapping Resource Offering.

2. Register user's Resource Offering at the Personal Profile Service instance using Discovery Service Modification.

3. Send Discovery Service Lookup request.

   Discovery service returns the discovery lookup response to the Web Service Consumer. The lookup responses contains the resource offering for the user's Personal Profile Service instance.

4. Send Data Service Query to the Personal Profile Service Instance to retrieve user attributes.

5. Send Data Service Modification to the Personal Profile Service Instance to modify user attributes.

To Run the Web Service Client Sample

1. Federate users `spUser` and `idpUser`.

   See "Verifying a Successful Liberty Setup" on page 36 for detailed instructions on federating user accounts, single sign-on, and single logout.

2. As `idpUser`, perform a single sign-on from the Service Provider to the Identity.

3. Using a browser, go to the following URL:

   `http://`*machine1*`:`*sever_port*`/wsc/index.jsp`

   You will see the boot strapping resource offering for the Discovery Service and two buttons: "Send Discovery Lookup" and "Add PP Resource Offering."

4. Click "Add PP Resource Offering."

   The `discovery-modify.jsp` page is displayed. The Personal Profile Service resource offering has been computed based on the bootstrapping Discovery Service Resource Offering.

5. Click "Send Discovery Update Request."

   The user's Personal Profile resource offering is registered in `idpUser` on *machine2*.

6. Click the "Return to index.jsp" link.

   The `index.jsp` page is displayed; you see the bootstrapping resource offering.

7. Click "Send Discovery Lookup."

   The `discovery-query.jsp` page is displayed.

8. (Optional) If you leave the "ServiceType to look for" field blank, all service instances will be returned.

   You can enter a value in the "ServiceType to look for" field. Values are defined in the Liberty Web Service Consumer specification. For example, `urn:liberty:idpp:2003-08` indicates a Personal Profile service.

9. Click "Send Discovery Lookup Request."

   The Personal Profile resource offering added in Step 4 will be displayed.

10. Choose one of the following two options:

    ❍ To query the Personal Profile Service in *machine2* for user attributes:

    a. Click "Send PP Query."

       The `id-sis-pp-query.jsp` page is displayed.

    b. In the Authentication Mechanism field, choose `urn:liberty:security:2003-08:null:null`.

    c. (Optional) You can change the value in the XPath Expression field to a different XPath expression for attribute selection. The default is `/PP/CommonName`.

    ❍ To modify a user's Personal Profile attributes:

    a. Choose one of the following:

       • Click “Send PP Modify.”

         The `id-sis-pp-modify.jsp page` is displayed and it sends a Modify request to the Personal Profile Service in *machine2*.

       • Click “Send PP Query.”

         The `id-sis-pp-query.jsp page` is displayed and it sends a Query request to the Personal Profile Service in *machine2*.

    b. In the Authentication Mechanism field, choose `urn:liberty:security:2003-08:null:null`.

    c. (Optional) You can modify the XPath Expression field for attribute selection. The default is `/PP/CommonName/AnalyzedName/FN`. Then modify the Value to include new values for the attribute.

Interacting with the Personal Profile Service

Follow these instructions to verify that you’ve successfully implemented the Personal Profile Service and can perform basic interactions with it.

1. Log in to Identity Server on *machine2* as Top-Level administrator. This is the Service Provider.
2. Create a policy for the Personal Profile service that requires user interaction for Query and Modify operations.

   a. In the console, click the Identity Management tab.

b. In the View menu, choose Policies, and then click New.

- In the Type menu, choose Normal policy.
- Enter the Name for this policy, and then click OK to create the policy.

c. In the View menu for this policy, choose Rules, and then click Add.

d. In the Service menu, choose "Liberty Personal Profile Service, " and then click "Next".

- In the Rule Name field, enter a name that will be meaningful to you.
- In the Resource Name field, enter an asterisk (*).
- Select the Modify or Query checkbox, or you can select them both.
- In the Value menu, select one of the following:

  Interact for Consent: Select this option if you want to ask the user for consent.
  Interact for Value: Select this option if you want to ask the user to provide values for the query response or the modify result.

- Click OK.
- Click Save to save the rule.

e. In the View menu for this policy, choose Subjects and then click Add.

- In the Type field, choose Authenticated Users, and then, click Next.
- In the Name field, enter a name for the subject.
- Click OK.
- Click Save to save the subject.

3. Enable policy evaluation for Personal Profile Service query and modify operations.

a. In the Identity Server console, click the Service Configuration tab.

b. In the Service menu, choose “Liberty Personal Profile Service.”

c. Check the two boxes labeled "is Query Policy Eval Required" and "is Modify Policy Eval Required."

d. Click Save to save the configuration.

4. To run the sample, follow the instructions in the section "Running the Web Service Consumer Sample" on page 42.

   You can change the policy defined in Step 2 to see different behavior for user interaction.

X.509 Message Authentication

The following sections provides steps setting up X.509 Message Authentication, and for verifying that you've authentication work properly in your Sample1 deployment

Setup

1. For *machine1* and *machine2*, follow the instructions in the SAML `xmlsig` sample to set up the JKS signing key store:

   is_install_root`/SUNWam/samples/saml/xmlsig`

2. Edit `/etc/opt/SUNWam/config/AMConfig.properties` to reflect the key store, password and cert alias.

3. In *machine1*, which is the Service Provider, locate the following file:

   `/etc/opt/SUNWam/config/AMConfig.properties`.

   Set the `com.sun.identity.liberty.ws.wsc.certalias` property to the alias of the signing certification.

To test X.509 Message Authentication in discovery service

1. Log in to Identity Server as Top-Level administrator.

2. Click the Service Configuration, then choose Discovery Service.

3. In the Discovery Service page, under "Resource Offering for Bootstrapping Resources,"click Edit.

4. In the Edit Resources Offering page, change the Authentication Mechanism from `urn:liberty:security:2003-08:null:null` to `urn:liberty:security:2003-08:null:X509`.

5. Click Save.

To test X.509 Message Authentication in Personal Profile Service,

1. Run the sample service. Follow the steps in the section "Running the Web Service Consumer Sample" on page 42.
2. Perform a Personal Profile service query or modify operation.

   Choose `urn:liberty:security:2003-08:null:X509` as the Authentication Mechanism.

To test SSL (urn:liberty:security:2003-08:TLS:X509),

1. Import the Ceritificate Authority (CA) for the Web Server on the Identity Provider (*machine2)* to the Web Server certificate database on the Service Provider (*machine1*).
2. Run the sample service.

   Follow the steps in the section "Running the Web Service Consumer Sample" on page 42.

Chapter 3

Federation Management

Once you've created a Liberty web service environment, you can modify service configurations or add new services to the system. This chapter provides instructions for modifying authentication domains, entity descriptors, and resource offerings. An Employee Profile Service sample is included to illustrate how you use the Federation APIs to create a new service and enable it for federation management.

Topics in this chapter include:

Overview of Authentication Domains and Providers

The Federation Management module provides an interface for creating, modifying, and deleting authentication domains, remote providers and hosted providers. The following steps demonstrate a basic Federation Management model:

1. Create an authentication domain.
2. Create one or more hosted providers that belong to the created authentication domain.

3. Create one or more remote providers that belong to the created 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.

The following sections explain how to create and configure authentication domains, remote providers, and hosted providers.

Managing Authentication Domains

This section describes how to create, modify, and delete authentication domains.

To Create An Authentication Domain

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

   The Create Authentication Domain is displayed in the Data pane.
3. In the Create Authentication Domain window, enter the name of the Authentication Domain.
4. Enter a value for the description of the Authentication Domain.
5. Enter a value for the Writer Service URL.

   Writer Service URL specifies the location of the Writer service that writes the cookie from the Common Domain. For example, if `example.com` is the common domain, the URL could be:

   `http://example.com:8080/common/writer`
6. Enter a value for the Reader Service URL.

   The Reader Service URL specifies the location of the service that reads the cookie from the Common Domain. For example, if `example.com` is the common domain, the URL could be:

   `http://example.com:8080/common/transfer`
7. Choose a status of active or inactive.

The default is active. This can be changed at any time during the life of the Authentication Domain by selecting the Properties icon. Choosing inactive disables Liberty communication within authentication domain, with respect to the current installation of Identity Server.

8. Click OK.

   The new Authentication Domain displays in the Navigation pane.

To Modify An Authentication Domain

1. Click on the Properties arrow next to the Authentication Domain you wish to modify.

   The properties of the Authentication Domain display in the Data pane.

2. Modify the properties of the Authentication Domain.

3. Click Save.

To Delete An Authentication Domain

Deleting an authentication domain does not delete the providers that belong to it. If providers belong to an authentication domain that has been deleted, they remain part of the authentication domain until they are explicitly removed. Additional providers can not be added to an authentication domain that has been deleted.

1. Choose Authentication Domains from the View menu in 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.

NOTE There is no warning message when performing a delete.

Managing Entity Descriptors

This section describes how to create, modify and delete entity descriptors. There are two types of entity descriptors you can define; a container entity (which can contain identity and service provider descriptors) and affiliate entities.

Once you have created an entity descriptor (of either type), you can add information for a contact person for that entity and a description of the organization to which the person belongs. The contact person is generally responsible for technical details concerning federation within his or her organization.

NOTE The contact person and organization concepts only exist within Federation Management, and have no correlation to the organization and user Identity Server object types.

Creating and Managing Providers

Creating a provider descriptor through the Identity Server console is a two-step process. First, you create an container entity. Next, you create the provider descriptor associated with that entity.

Once the provider descriptor is defined, you can modify any of the provider's attributes by selecting Provider from the View menu located in the Navigation pane.

To Create a Container Entity

1. Choose Entity Descriptors from the View menu in the Federation Management module.
2. Click New. The Create Provider window is displayed.
3. Enter a value for the Entity ID.

   The Entity ID should specify the URL identifier of the entity. It must be unique across all entities.
4. Enter a description of the entity.
5. Select Provider from the Type option.
6. Click OK.

To Create and Manage a Provider Descriptor

1. Choose Entity Descriptors from the View menu in the Federation Management module.
2. Select the Entity that will contain the provider.

3. Select Provider from the View menu in the Navigation pane.
4. Click the New Provider button to display the first page of the New Provider Wizard.
5. Select the provider type and enter information into the common provider attribute fields. The fields are as follows:

   Provider ID. The Provider ID should specify the URL identifier of the provider. It must be unique across all remote and hosted providers.

   Description. Enter a description of the provider.

   Provider is of Type Identity. Decide if the provider is to be defined as an identity provider. By default, all providers are service providers. If selected, the Identity Provider option will additionally define the provider as an identity provider.

   Provider is Hosted (Local). If selected, the provider is a hosted provider. By default (not selected), the provider is a remote provider.

   Valid Until. This field allows you to enter the expiration date for the metadata pertaining to the provider. Use the following format:

   `yyyy-mm-dd hh:mm:ss.SZ`

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

   Cache Duration. This field defines the duration period for the metadata to be cached and uses the `xs:duration` format.

   Protocol Support Enum. This field defines the protocol release supported by the entity. `urn:liberty:iff:2003-08` refers to Identity Federation Framework (ID-FF) 1.2 and `urn:liberty:iff:2002-12` refers to Federation Identity Framework (ID-FF) 1.1.

   Security Key. The Security Key defines the Security Certificate alias. The certificates are stored in the JKS keystore against an alias. This alias (the Security Key) is used to fetch the required certificate.

   Key Use.This field defines allowed key usage. You can choose `encryption` or `signing`.

   Key Size. This field constrains the length of keys used by the consumer when interacting with another entity.

   Encryption Method. This field defines the encryption preferences URI.

   Server Name Identifier Mapping Binding. This field defines the SAML authority binding at the identity provider to which identifier mapping queries

are sent.

Additional Meta Locations. This field specifies the location of other relevant metedata about the provider.

6. Click Next.

7. Enter the information for the communications and service provider attributes. The fields are as follows:

Communication URLs

SOAP Endpoint URL. This field specifies the location for the receiver of SOAP requests. This is used to communicate on the back-channel (non-browser communication) through SOAP.

Single Logout Service URL. The Single Logout Service URL is used by a service provider or identity provider to send and receive logout requests.

Single Logout Return URL. This specifies the URL to which logout requests are redirected after processing.

Federation Termination Service URL. This field specifies the URL to which federation termination requests are sent.

Federation Termination Return URL. This field specifies the URL to which federation termination requests are redirected after processing.

Name Registration Service URL. This field uses the Name Registration protocol that is used by a service provider to register its own Name Identifier while communicating to an identity provider. Registration occurs only after a federation session is established. This field defines the service URL used by a service provider to register a Name Identifier with an identity provider.

Name Registration Return URL. This field uses the Name Registration protocol that is used by a service provider to register its own Name Identifier while communicating to an identity provider. Registration occurs only after a federation session is established. The Name Registration Return URL is the URL to which the identity provider sends back the status of the registration.

Communication Profiles

Federation Termination Profile. You can choose SOAP or HTTP / Redirect. This field specifies if the SOAP or HTTP / Redirect profile is to be used to notify of federation termination. This can be changed at any time during the life of the provider.

Single Logout Profile. You can choose SOAP or HTTP Redirect. This field specifies if SOAP or HTTP Redirect is to be used to notify a logout event. This

can be changed at any time during the life of the provider.

Name Registration Profile. You can choose SOAP or HTTP / Redirect. This field specifies if the SOAP or HTTP / Redirect profile is to be used for name registration. This can be changed at any time during the life of the provider.

Server Relationship Term Notification URL. This field defines a URI describing the profiles that the entity supports for relationship termination.

Single Sign-on/Federation Profile. This field specifies the profile used by the hosted provider for sending authentication requests. Identity Server provides the following protocols:

- Browser Post - specifies a front-channel (http POST-based) protocol.
- Browser Artifact - Backchannel (non-browser) SOAP-based protocol.
- LECP - Liberty Enabled Client Proxy.

Assertion Consumer URL. This field defines the provider end-point to which a provider will send SAML assertions.

Assertion Consumer URL ID. This ID is required if Protocol Support Enum is urn:liberty:iff:2002-12.

Set Assertion Consumer Service URL as Default. This option sets the Assertion Consumer URL as the default.

Sign Authentication Request. This option, if enabled, specifies that the provider send signed authentication and federation requests. The identity provider will not process unsigned requests originated from the service provider.

Name Registration After Federation. If enabled, this option allows for a service provider to participate in name registration after it has been federated. 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.

8. If the provider you are creating is a local provider, enter data for the Identity Server Configuration. If the provider is not local, skip this step. The fields are:

Provider URL. This field defines the URL of the local provider.

Alias. This field allows you to enter an alias name for the local provider.

Authentication Type. Remote / Local - This field specifies if the hosted provider should contact an identity provider for authentication upon receiving an authentication request (Remote), or if authentication should be done by the

hosted provider itself (Local).

Default Authentication Context. This field specifies 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 default values are:

- Previous-Session
- Time-Sync-Token
- Smartcard
- MobileUnregistered
- Smartcard-PKI
- MobileContract
- Password
- Password-ProtectedTransport
- MobileDigitalID
- Software-PKI

Force Authentication at Identity Provider. This option indicates if the identity provider must reauthenticate (even during a live session) when an authentication request is received.

Request Identity Provider to be Passive. If selected, this option specifies that the identity provider must not interact with the principal and must interact with the user

Organization DN. This field specifies the storage location of the DN of the organization if each hosted provider chooses to manage users across different organizations leading to a hosted model.

Liberty Version URI. This field specifies the version of the Liberty specification.

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.

Provider Home Page URL. This field specifies the home page of the provider.

Single Sign-on Failure Redirect URL. This field specifies the home page of the

provider.

Assertion Interval. This field specifies the validity interval for the assertion issued by an identity provider. A principal will remain authenticated by the identity provider until the assertion interval expires.

Cleanup Interval. This field specifies the interval of time to clear assertions that are stored in the identity provider.

Artifact Timeout. This field specifies the timeout of a identity provider for assertion artifacts.

Assertion Limit. This field specifies the number of assertions an identity provider can issue, or the number of assertions that can be stored.

9. Click Next.

   Enter the values for the organization and contact person. For more information, see To Add a Contact Person and Organization.

10. Click Next.

11. Select the authentication domains to which the provider will belong.

    Use the direction arrows to move a selected authentication domain into the Available list. Click Save. This will assign the provider to the authentication domain. A provider can belong to one or more authentication domains, however a provider without any authentication domains specified can not participate in Liberty communications. Click Save.

12. Click Finish.

Creating and Managing Affiliates

In the Identity Server console, you create an affiliate through the New Entity page. Once the affiliate descriptor is defined, you can modify any of the affiliate's attributes and manage the members of the affiliation by selecting Affiliates from the View menu located in the Navigation pane.

To Create an Affiliate Entity

1. Choose Entity Descriptors from the View menu in the Federation Management module.
2. Click New.
3. Enter a value for the Entity ID.

The Entity ID should specify the URL identifier of the entity. It must be unique across all entities.

4. Enter a description of the entity.
5. Select Affiliate from the Type option.
6. Enter the Affiliate ID.

   The Affiliate ID should specify the URL identifier of the affiliate. It must be unique across all entities. This field is required.

7. Enter the Affiliate Owner ID.

   The Affiliate OwnerID is the Provider ID of the owner or parent operator of the affiliate, from which additional metadata can be received. This field is required.

8. Click OK.

To Manage an Affiliate Descriptor

1. Choose Entity Descriptors from the View menu in the Federation Management module.
2. Select the Entity that will contain the Affiliate.
3. Select Affiliates from the View menu in the Navigation pane.
4. Modify the affiliate attribute fields. The fields are as follows:

   Valid Until. This field allows you to enter the expiration date for the metadata pertaining to the affiliate in the following format:

   `yyyy-mm-dd hh:mm:ss.SZ`

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

   Cache Duration. This field defines the duration period for the metadata to be cached and uses the `xs:duration` format.

   Security Key. The Security Key defines the Security Certificate alias. The certificates are stored in the JKS keystore against an alias. This alias (the Security Key) is used to fetch the required certificate.

   Key Use.This field defines allowed key usage. You can choose `encryption` or `signing`.

   Key Size. This field constrains the length of keys used by the consumer when interacting with another entity.

Encryption Method. This field defines the encryption preferences URI.

Affiliate Members. This field allows you to define one or more providers that will be members of the affiliation. The providers that are displayed are pre-defined in existing container entity descriptors.

Use the direction arrows to move a selected provider into the Available list. Click Save. This will assign the provider to the authentication domain. A provider can belong to one or more affiliates.

5. Click Save.

To Add a Contact Person and Organization

1. Choose Entity Descriptors from the View menu in the Federation Management module.
2. Click the Properties arrow next to the entity you wish to modify.
3. In the Data pane, choose General from the View menu (this is the default view).
4. Enter a general description of the contact person and organization.
5. Enter a date in the Valid Until field.

   This field allows you to enter the expiration date for the metadata pertaining to the contact person. Use the following format:

   `yyyy-mm-dd hh:mm:ss.SZ`

   For example, `2004-12-31 12:30:00.0-0800`
6. Enter a value for the Cache Duration.

   This field defines the duration period for the metadata to be cached and uses the `xs:duration` format.
7. Enter the following fields in the Contact Person section:

   First Name. The first name of the contact person.

   Last Name. The last name of the contact person.

   Type. The contact type. This can be one of the following:

   - Technical
   - Administrative

- Billing
- Other

Company. The contact person's company name.

Liberty Principal Identifier. The name identifier that points to an online instance of the contact person's personal information profile (PIP).

Email. The email address of the contact person.

Telephone. The telephone number of the contact person.

8. Enter the following fields in the Organization section (all fields are mandatory):

   Name. The name of the entity's organization.

   Display Name. The name of the organization that is displayed to the principal.

   URL. The URL of the organization. This URL is used by a user agent to direct a principal for additional information on the entity.

9. Click Save.

Deleting Entity Descriptors

1. Choose Entity Descriptors from the View menu in Federation Management.

   All created entities display in the Navigation pane.

2. Check the boxes of the entity descriptor you want to delete.
3. Click Delete.

NOTE There is no warning message when performing a delete.

Managing Resource Offerings

A resource offering is the association of a resource and a service instance. Typically, a single service instance will serve many resources. For example, a personal profile service provider will serve multiple profiles to a single service instance. It would be impractical to have a separate protocol endpoint for each profile.

The Discovery Service is an identity service that allows requestors to discover resource offerings. In Identity Server, resource offerings can be stored and managed in three different ways:

- User Discovery Resource Offering - This is a resource offering associated with particular user. To access the User Discovery Resource offering, select Users in the Identity Management module, and choose Resource Offerings from the View menu in the Navigation pane.
- Dynamic Resource Offering - This is a resource offering associated with and organization or role.
- Resource Offering for Bootstrapping Mode - This resource offering is accessed through the Discovery Service in the Service Configuration module. The resource offering is sent to the service provider or web service client (WSC) in the Single Sign-on assertion during Single Sign-on.

To Define Resource Offering

NOTE Steps 1 and 2 only apply to User Discover Resource Offering.

1. Enter a value for the Resource ID Attribute.

   This field defines an identifier for a Resource ID value.

2. Enter the Resource ID Value.

   This field defines the URI used to identify a particular resource. It must not be a relative URI and should contain a domain name which is owned by the provider that is hosting the resource. If a resource is exposed through multiple Resource Offering elements, all of those resource offering elements should have the same Resource ID value.

   Example of a Resource ID value:

   ```
   http://profile-provider.com/profiles/14m0B82k15csaUxs
   ```

   ```
   urn:libery:isf:implied-resource
   ```

NOTE The following steps apply to all resource offering mechanisms.

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

4. Enter the Service Instance.

   This field defines an active web service at a distinct endpoint.

5. Enter the Service Type.

   This field contains the URI that defines the type of service that service instance implements. For example:

   ```
   urn:liberty:id-sis-pp:2003-08
   ```

6. Enter the Provider ID.

   This field contains the URI of the provider of this service instance. For example:

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

7. Define the Service Description.

   For each resource offering profile, at least one service description must be defined. The service description fields are as follows:

   Security Mechanism ID. This field lists all available security mechanisms that the service instance supports, which define how a web service client authenticates to the web service provider. 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 and Move Down buttons.

   Brief SoapHttp Description. If selected (default), this provides inline the information necessary to invoke basic SOAP-over-HTTP-based service instances, without using Web Service Description Language (WSDL).

   End Point. This field contains the URI of the SOAP-over-HTTP endpoint. The URI scheme must be HTTP or HTTPS. For example:

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

   SOAP Action. This field contains the equivalent of the `wsdlsoap:soapAction` attribute of the `wsdlsoap:operation` element in a WSDL-based description.

   WSDL Reference. This field references an external concrete WSDL resource.

   Service Namespace. 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.

   Service Local Part. This field provides the local part of the qualified name of the service namespace URI.

8. Enter the Resource Offering Options.

   This field lists the options available for the resource offering to provide hints to a potential requestor whether certain data or operations are available to a particular offering. If no option is specified, the service instance does not advertise any available options.

9. Choose the resource offering Directives. The directives are as follows:

   Authorize Requester. This directive is specified for discovery service provider to include a SAML assertion containing a `ResourceAccessStatement` in any future `QueryResponse` message.

   Authenticate Session Context. This directive is specified for discovery service provider to include a SAML assertion containing a `SessionContextStatement` in any future `QueryResponse` message.

   AuthenticateRequester. This directive must be used with any descriptions, including the security mechanisms from `LibertySecMech` which uses SAML for message authentication.

   EncryptResourceID. This directive specifies that the discovery service must not reveal the unencrypted resource ID to the clients. Currently, this directive is not supported, so the resource ID will not be encrypted when this directive is selected. If you wish to associate a directive with one or more description elements in the resource offering, select the checkbox in front of that Description ID. If none of the Description IDs are selected, the directive is applied to all description elements provided in the resource offering.

10. Click Save.

Adding a New Liberty Web Service

Adding a Liberty web service to Identity Server entails the following:

- Developing the Server-Side Code
- Configuring the Service Schema
- Setting Up the Back-End Data Store
- Deploying the Service on the Identity Provider
- Deploying the Client on the Service Provider
- Running the Web Service Client

Sample code for an Employee Profile service is provided with Identity Server to illustrate how to implement and deploy a new data service instance in Identity Server. The sample implements a Liberty Employee Profile service.

An Employee Profile Service Example

For deploying and running of this sample, you will need two Identity Server installations. One serves as Liberty Service Provider (SP), and one serves as Liberty Identity Provider (IDP). The Employee Profile service will be located in Identity Provider, and client code in the forms of `jsp` files will be located in SP.

Table 3-1 Two machines required for this sample

Variable Placeholder	Host Name	Deployed on this host
machine1	`www.sp1.com`	• Service Provider • Web Service Client (.jsp files)
machine2	`www.idp1.com`	• Identity Provider • Discovery Service • Employee Profile Service

The sample provides five .jsp files.

Table 3-2 JSPs provided in this sample

Name	Description
`index.jsp`	Retrieves boot strapping resource offering for discovery service.
`discovery-modify.jsp`	Adds Resource Offering for a user.

Table 3-2 JSPs provided in this sample

Name	Description
`discovery-query.jsp`	Sends query to discovery service for service resource offering.
`id-sis-ep-modify.jsp`	Sends Data Service Modify request to modify user attributes.
`id-sis-ep-query.jsp`	Sends Data Service Query Request to retrieve user attributes

Developing the Server-Side Code

Since the Employee Profile service is a data service, when you develop its server-side code, you must extend `DSTRequestHandler` and implement its `processDSTRequest()` method to process query and modify requests.

Identity Server provides sample code to demonstrate how you would develop and deploy a new service. The Employee Profile service sample code is already written and resides in the following directory:

IdentityServer_base`/samples/phase2/sis-ep/src/ep`

NOTE This sample does not provide instructions for implementing every aspect of a complete service. For this example, note the following:

- Authorization is not enabled.
- The only authentication mechanism supported is `urn:liberty:security:2003-08:null:null`
- SSL/X509 and SAML tokens are not supported.
- Uses simple-minded select string parsing instead of using XPATH API
- A few attributes such as `LInternalJobTitle`, `LOU`, `LCN`, `LAltCN` and `LLegalName` not supported.

Identity Server's back-end data store is used in this sample to store the requested Employee Profile data. The code for getting and setting the data is included in the Service Management APIs. See “Defining A Custom Service” in the *Identity Server Developer’s Guide*.

Configuring the Service Schema

The `xsd` file which defines the Employee Profile service schema is the starting point for developing the Employee Profile service server-side code.

1. Invoke the jaxb compiler on xsd files.

   The Employee Profile service and the related schema files are gathered under *IdentityServer_base* / SUNWam / samples / phase2 / sis-ep / xsd.

   a. Be sure that the jaxb compiler xjc is available somewhere in your environment. If it is not available, download JWSDP 1.3 from Sun's web site.

      http://java.sun.com

   b. To make sure your are using the correct xjc.sh path inside the script invoke_xjc.sh (which is under *IdentityServer_base* / SUNWam / samples / phase2 / sis-ep / bin), run the following command:

      ```
      # jar tvf install_dir/SUNWam/lib/am_services.jar | grep
          "impl/runtime"
      ```

      Based on the result, in the script invoke_xjc.sh, replace the package name behind the option -use-runtime.

   c. Be sure that JAVA_HOME environment variable in xjc.sh is set; it should point to a JDK version equal to or higher than 1.4.

   d. Invoke the script invoke_xjc.sh:

      ```
      # IdentityServer_base / SUNWam / samples / phase2 / sis-ep/bin/
          invoke_xjc.sh
      ```

   A number of Java files are generated from the xsd files and are placed under *IdentityServer_base* / SUNWam / samples / phase2 / sis-ep/xsd/gen.

2. Compile the generated Java files.

   a. If necessary, modify the INSTALL_DIR variable in script *IdentityServer_base* / SUNWam / samples / phase2 / sis-ep/ bin/compile_gen.sh.

   b. Run the following command:

      ```
      # IdentityServer_base / SUNWam / samples / phase2 / sis-ep/
          bin/compile_gen.sh
      ```

   A number of class files are generated and placed under *IdentityServer_base* / SUNWam/samples/phase2/sis-ep/classes. The *IdentityServer_base*/SUNWam/samples/phase2/sis-ep/classes directory is soft-linked from *install_dir*/SUNWam/lib which is already on the web container's class path.

3. Develop the Employee Profile service code. (See the introduction to "Developing the Server-Side Code" on page 65.)

4. Compile the EP service code.

   ```
   # cd IdentityServer_base/SUNWam/samples/phase2/sis-ep/src
   # make
   ```

   The generated class files are placed under *IdentityServer_base*/SUNWam/samples/phase2/sis-ep/classes, together with the class files compiled from the generated jaxb Java files.

5. Set up the back-end data store.

   See the following section.

Setting Up the Back-End Data Store

For more information about the back end data store and service management, see the *Identity Server Administration Guide.*

To Set up the Back-End Data Store

1. Load the Employee Profile LDIF file. This is the Directory Server schema.

   a. # cd *IdentityServer_base*/SUNWam/samples/phase2/sis-ep/bin

   b. Modify load_ldif.sh.

      Provide the values that are appropriate for your deployment for the following parameters: installation directory, host name, directory server port and password.

   c. Run the load_ldif.sh script:

      ```
      # ./load_ldif.sh
      ```

      This loads the Directory Server schema defined in *IdentityServer_base*/SUNWam/samples/phase2/sis-ep/ldif/ep.ldif into the Directory Server. The attribute names used in ep.ldif are the ones used in EmployeeProfile.java in *IdentityServer_base*/SUNWam/samples/phase2/sis-ep/src/ep.

2. Load the Employee Profile service management schema.

a. In *IdentityServer_base* / SUNWam / samples / phase2 / sis-ep/ bin, modify `load_xml.sh`.

Change the `amadmin` password. Change the installation directory, if necessary.

b. Run the following command:

```
# ./load_xml.sh
```

This loads the Identity Server service management schema that is defined in *IdentityServer_base* / SUNWam / samples / phase2 / sis-ep/ `xml/amLibertyEmployeeProfile.xml` into the Identity Server and ultimately into Directory Server.

Note that the attribute names in `amLibertyEmployeeProfile.xml` are the same as those in `ep.ldif`.

Deploying the Service on the Identity Provider

To Deploy the Service

1. Configure the SOAP Receiver to recognize `EPRequestHandler`.

a. Log in to Identity Server as Top-Level administrator.

b. Click the Service Configuration tab, and then select SOAP Binding.

c. In the right pane, add `key=idep|class=ep.EPRequestHandler` to the RequestHandler List.

Note that `ep.EPRequestHandler` is the class name for *IdentityServer_base* / SUNWam / samples / phase2 / sis-ep/ `src/ep/EPRequestHandler.java` which extends `DSTRequestHandler`. Also note that `idep` will be part of the URI used to invoke the Employee Profile service.

d. In `/etc/opt/SUNWam/config/AMConfig.properties`, add `ep.jaxb` to the property `com.sun.identity.liberty.ws.jaxb.packageList`.

This is to let the SOAP binding layer knows about the Employee Profile service `jaxb` package which is new to the Identity Server platform.

2. Register the Employee Profile service to the default organization.

a. Log in to Identity Server As Top-Level administrator.

b. In the Identity Management tab, in the View menu, choose Services.

c. Click Add.

d. In the right pane, select "Liberty Employee Profile Service," and then click OK.

3. Create a user. For this example, name the user `idpUser`.

   Select "Liberty Employee Profile Service" in the "Available Services" list creating `idpUser` (otherwise EP modify will fail).

   If `idpUser` exists already (from deploying the Web Service Client sample, for example), then just add the "Liberty Employee Profile Service" for this user.

   This user will be used as the federated user on the Identity Provider side, also for storage of Discovery Service resource offering and Employee Profile service attributes. Deploy liberty sample1 IDP to set up a runnable liberty scenario.

4. Deploy the Identity Provider.

   Follow the instructions in the section "Deploying the Identity Provider" on page 33. If this is done already, when running the Web Service Client sample, for example, then skip this step.

5. Restart the web container in which Identity Server is running.

Deploying the Client on the Service Provider

To Deploy the Client

1. Deploy the Service Provider.

   Follow the instruction in the section "Deploying the Identity Provider" on page 33. If this is already done, for example if you have already run the Web Service Client sample, then skip this step.

2. Change protocol support of the remote Identity Provider to ID-FF 1.2.

   a. Log in to Identity Server as Top-Level administrator.

   b. Click the Federation Management tab, and then in the View menu choose Entity Descriptors.

   c. Click the `remoteIDP` entity ID from the list.

   d. In the right pane, in the view menu, choose Provider.

   e. Click the Edit link under Provider.

f. In the "Protocol Support Enum" field, choose `urn:liberty:iff:2003-08`.

g. Click Save.

3. Replace the tags and hosts in `discovery-modify.jsp` and `index.jsp`.

All the JSP files are under *IdentityServer_base*/SUNWam/samples/phase2/sis-ep/jsp/. Use the Data Services Template client API and Discovery client API for sending query or modify requests and for receiving query or modify responses.

- Replace `IDP_SERVER_PORT` with the server port of Identity Provider host.
- Replace `SERVICE_DEPLOY_URI` with the service deployment URI of the Identity Provider host.
- Replace `www.sp1.com` with host name of the Service Provider host if necessary.
- Replace `www.idp1.com` with host name of Identity Provider host if necessary.

4. Deploy the JSP files.

Copy all the five JSP files to a sub directory of the document root of the web container.

For example, if you are using Sun Java System Web Server 6.1, run following command:

```
# mkdir webserber_install_dir/docs/wsc
# cp is_install_dir/SUNWam/samples/phase2/ep/*.jsp
    webserber_install_dir/docs/ep/
```

5. Create a user named `spUser`.

This user will be used as federated user on the Service Provider side.

6. Restart the web container in which Identity Server is running.

Running the Web Service Client

Before you can run the Web Service Client, you must deploy a Service Provider and an Identity Provider. If you have not done this yet, see “Creating a Liberty Web Services Environment” on page 29 for detailed instructions.

1. Federate users `spUser` and `idpUser`.

   See “Creating a Liberty Web Services Environment” on page 29. Follow the instructions for federating user accounts, performing single sign-on, and performing single logout.

2. As `idpUser`, perform a single sign-on again from the Service Provider to the Identity Provider.

3. In a browser, connect to `http://`*machine1*`:`*sever_port*`/ep/index.jsp`.

   You will see the bootstrapping resource offering for Discovery Service and two two buttons: "Send Discovery Lookup" and "Add PP Resource Offering."

4. Click "Add PP Resource Offering."

   The `discovery-modify.jsp` page is displayed. The Employee Service resource offering has been computed based on the bootstrapped Discovery Service Resource Offering.

5. Click "Send Discovery Update Request.",

   The user's Employee Profile resource offering will be registered in `idpUser` on *machine2*.

6. Click the "Return to index.jsp" link.

   This will bring you back to the `index.jsp` page with the bootstrapped resource offering.

7. Click "Send Discovery Lookup."

   The `discovery-query.jsp page` is displayed.

8. (Optional) If you leave the “ServiceType to look for” field blank, all service instances will be returned.

   You can enter a value in the "ServiceType to look for" field. Values are defined in the Liberty Web Service Consumer specification.

9. Click "Send Discovery Lookup Request."

   The Employee Profile resource offering added in Step 4 will be displayed. Choose one of the following two options:

- To query Employee Profile Service in *machine2* for user attributes, click "Send EP Query."

  The id-sis-ep-query.jsp page is displayed.

  In the Authentication Mechanism field, choose `urn:liberty:security:2003-08:null:null`. You can change the "XPath Expression" field to a different XPath expression for attribute selection. The default is `/EP/EmployeeID`.

- To modify the user's employee profile attributes, click "Send PP Modify." The `id-sis-ep-modify.jsp` page is displayed. The modify request is sent to the Employee Profile Service in *machine2*.

  In the Authentication Mechanism field, choose `urn:liberty:security:2003-08:null:null`. You can modify the "XPath Expression" field for attribute selection. The default is `/EP/EmployeeID`. You can modify the Value field with new values for the attribute.

Constructing a PAOS Request and Response

Identity Server provides client PAOS APIs and a sample stand-alone Java application to demonstrate how to set up and invoke a PAOS service interaction between a client and a server. In a real-world deployment, the server-side code would be developed by a service provider.

All the files required to use the sample are located here:

IdentityServer_base`/SUNWam/samples/phase2/paos`

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 client or consumer.

To Run the Sample PAOS Program

1. Set the environment variables.
   - These environment variables will be used for running the `make` and `gmake` commands, and for using the `run` command. The `Makefile` and `run` command are in the same directory where the sample files are located:

     IdentitySever_base`/SUNWam/samples/phase2/paos`
   - Set the `JAVA_HOME` variable to your installation of JDK. The JDK version should be newer than JDK 1.3.1
2. In the directory `IdentityServer_base/SUNWam/samples/phase2/paos`, run `gmake` or `make`.
3. Copy the `PAOSClientServlet.class` file to the *IdentityServer_base*`/SUNWam/lib` directory. Alternatively, you can make a soft-link in that directory back to this class.
4. Register the Sample servlet.

   In the file *IdentityServer_base*`/SUNWam/web-apps/services/WEB-INF/web.xml`, insert these lines just after the last */servlet* tag:

```
<servlet>
          <servlet-name>PAOSClientServlet</servlet-name>
          <description>PAOSClientServlet</description>
          <servlet-class>PAOSClientServlet</servlet-class>
      </servlet>
```

   Insert the following lines just after the last `</servlet-mapping>` tag:

```
<servlet-mapping>
          <servlet-name>PAOSClientServlet</servlet-name>
          <url-pattern>/PAOSClientServlet</url-pattern>
      </servlet-mapping>
```

5. Restart the server.
6. Modify the `run` script to reflect your particular installation.

7. Run `PAOSServer`:

   ```
   ./run
   ```

You will see the output from the `PAOSServer` program. You can also see the output from `PAOSClientServlet` program in the Web Server's log file. For example, for Sun Java System Web Server, in the Web Server instance directory, look in the `log` subdirectory for the `errors` file.

Chapter 4

Service Configuration Attributes

This chapter provide summaries of service configuration attributes that come with Identity Server. The chapter contains the following topics:

Discovery Service Attributes

The Discovery Service attributes are global attributes. The values applied to them are applied across the Identity Server configuration and are inherited by every configured organization. (They cannot be applied directly to roles or organizations, as the goal of global attributes is to customize the Identity Server application). The Discovery Service attributes are:

- Provider ID
- Supported Authentication Mechanisms
- Supported Directives
- Do Policy Evaluation for DiscoveryLookup
- Do Policy Evaluation for DiscoveryUpdate
- Class for Authorizer Plugin
- Class for Discovery Service Entry Handler Plugin
- Classes For Resource ID Mapper Plugin
- Generate Session Context Statement for Bootstrapping

- Resource Offerings for Bootstrapping

Provider ID

This attribute defines the unique identifier used for this Discovery Service. For example:

```
http://example.com:58080/amserver/Liberty/disco
```

Supported Authentication Mechanisms

This attribute specifies the authentication mechanisms supported by the Discovery Service. By default, all of the mechanisms are selected. If an authentication mechanism is not selected, and a WSC sends a request using that authentication mechanism, the request will be rejected without passing it to the corresponding WSP.

Supported Directives

This attribute allows you to select the directives that are supported by the Discovery Service. If a service provider wants to insert an entry with an unsupported directive, the request will fail.

Do Policy Evaluation for DiscoveryLookup

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

Do Policy Evaluation for DiscoveryUpdate

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

Class for Authorizer Plugin

This attribute defines the classname and classpath used for policy evaluation.

Class for Discovery Service Entry Handler Plugin

This attribute defines the classname and classpath used to set or retrieve `DiscoEntries`.

Classes For Resource ID Mapper Plugin

This attribute contains a list of entries that are used to generate the Resource ID for a resource offering configured for an organization or role. The entries contain a key / value pair (separated by " | ") in the following format:

`providerID=`*`providerID`*`|`*`classname_classpath`*

To add a new request handler, click the add button. The key and value parameters are required.

Generate Session Context Statement for Bootstrapping

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

Resource Offerings for Bootstrapping

This attribute defines the service's resource offering for bootstrapping. 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. If you have not defined a resource offering, click New. If you wish to edit an existing resource offering, click the Edit link. For more information defining a resource offering, see "Managing Resource Offerings" on page 60.

Liberty Personal Profile Service Attributes

The Liberty Personal Profile service attributes are global attributes. The values applied to them are applied across the Sun Java System Identity Server configuration and are inherited by every configured organization. (They can not be applied directly to roles or organizations as the goal of global attributes is to customize the Identity Server application.)

The Liberty Personal Profile Service Attributes are:

- Resource ID 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

Resource ID Mapper

This attribute specifies the mutual implementation of a resourceID to the User DN.

Authorizer

This attribute defines the default implementation of the Personal Profile Service service authorization.

Attribute Mapper

This attribute defines the mapping between a Liberty Personal Profile service attribute to a user attribute. Format:

LibertyPersonalProfileAttribute=IdentityServerAttribute

For example:

```
AltCN=SunIdentityServerPPCommonNameAltCN
```

Provider ID

This attribute defines the unique identifier used for this Liberty Personal Profile Service. For example:

```
http://example.com:58080/amserver/Liberty/idpp
```

Name Scheme

This attribute defines the naming scheme that will be used for the Liberty Personal Profile Service common name. For example, you can specify first and last name, or first, middle and last name.

Namespace Prefix

This attribute specifies the namespace prefix to be used for Liberty Personal Profile Service XML protocol messages. NameSpace is used to differentiate the elements that come from different XML schemas. Namespace prefix is a prefix to the element and will be useful to define XML metadata from different XML schema namespaces.

Supported Containers

This attribute defines the list of supported Personal Profile containers. To add a container, click the Add button. Enter the key value pair in the provided fields and click OK.

PPLDAP Attribute Map List

This attribute list specifies the mapping for the Personal Profile attributes defined in the Liberty II specification to the Identity Server Personal Profile service attributes.

For example, in the mapping scheme, `JobTitle=sunIdentityServerPPEmploymentIdentityJobTitle`, `sunIdentityServerPPEmploymentIdentityJobTitle` is the Identity Server attribute that maps to the Liberty Protocol's `JobTitle` attribute.

Require Query PolicyEval

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

Require Modify PolicyEval

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

Extension Container Attributes

This attribute specifies the list of extension container attributes for the Personal Profile service.

Extension Attributes Namespace Prefix

This attribute defines the namespace prefix for the extensions defined in Extension Container Attributes.

SOAP Binding Service Attributes

The SOAP Binding Service attributes are global attributes. The values applied to them are carried across the Sun Java System Identity Server configuration and inherited by every configured organization. (They can not be applied directly to roles or organizations as the goal of global attributes is to customize the Identity Server application.)

The SOAP Binding Service attributes are as follows:

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

Request Handler List

This attribute stores information about a Web Service Provider (WSP) deployed in Identity Server. It lists entries that contain a key / value pair (separated by " | "). For example:

```
key=disco|class=com.example.identity.liberty.ws.disco.DiscoveryService|soa
pActions=sa1 sa2 sa2
```

To add a new request handler, click the add button. The key and class parameters are required. The parameters are:

key. This defines the second part of the URI path for the SOAP endpoint of the WSP. The first part is defined as Liberty by the SOAP services. For example, if you define `disco` as the key, the SOAP endpoint for the Discovery service is:

protocol`://`*hostname:port/deloy_uri*`/Liberty/disco`

class. This parameter specifies the name of the implementation class for the WSP. The Liberty SOAP layer provides a handler interface to be implemented by each WSP to process the requested message and then return a response.

soapActions. This is an optional parameter that specifies supported SOAPActions. If this parameter is not specified, all SOAPActions are supported. If a Web Service Consumer (WSC) sends a request with an unsupported SOAPAction, the request will be rejected by the SOAP layer without passing it one to the corresponding WSP.

Web Service Authenticator

This attribute defines the implementation class for the `WebServiceAuthenicator` interface, which authenticates and generates a credential for a Web Service Consumer (WSC), based on the request.

Supported Authentication Mechanisms

This attribute specifies the authentication mechanisms supported by the SOAP endpoint. By default, all of the mechanisms are selected. If an authentication mechanism is not selected, and a WSC sends a request using that authentication mechanism, the request will be rejected by the SOAP layer without passing it to the corresponding WSP.

Chapter 5

Using the Web Services Client APIs

Sun ™ Java System Identity Server 2004Q2 provides a Federation Management framework for creating, discovering, and consuming identity services. This chapter provides engineering notes and summaries of Java classes and APIs that you can use to extend the framework. The SOAP-based invocation framework includes the following components and core identity services:

Federation Packages and Global Interfaces

Table Table 5-1 summarizes the public APIs you can use to deploy Liberty II components or extend the core services. For detailed API reference that includes classes, methods and their syntax and parameters, see the Javadocs at `/opt/SUNWam/docs/am_public_javadocs.jar`.

Table 5-1 Summary of Liberty II Web Services Packages

Package Name	Description
`com.sun.identity.liberty.ws.common.wsse`	Provides an interface for Web Service Security X.509 Certificate Token Profile. See "Authorization" on page 88.
`com.sun.identity.liberty.ws.disco`	Provides interfaces to manage liberty discovery service. See "Discovery Service" on page 89.
`com.sun.identity.liberty.ws.disco.plugins`	Provides interfaces to manage liberty discovery service. See "Discovery Service" on page 89.
`com.sun.identity.liberty.ws.dst`	Provides interface to manage liberty ID-WSF Data Service Template. See "Data Services Template" on page 91.
`com.sun.identity.liberty.ws.interaction`	Provides classes to support Liberty Interaction RequestRedirect Profile. See "Interaction Service" on page 97.
`com.sun.identity.liberty.ws.interfaces`	Provides interfaces common to all liberty services.
`com.sun.identity.liberty.ws.paos`	Provides classes for web application to construct and process PAOS request and response."PAOS" on page 98.
`com.sun.identity.liberty.ws.security`	Provides interface to manage liberty ID-WSF security mechanisms."Security Token Manager" on page 85.

Table 5-2 summarizes classes used by all Liberty service components, and interfaces common to all Liberty services. All of these are Included in package `com.sun.identity.liberty.ws.common`.

Table 5-2 Common Liberty II Classes and Interfaces

Class or Interface Name	Description
`LogUtil`	Included in package com.sun.identity.liberty.ws.common.
`Status`	Class that represents a common status object. Included in package com.sun.identity.liberty.ws.common.
`Authorizer`	Interface for identity service to check authorization of a WSC. Included in package `com.sun.identity.liberty.ws.common.interfaces`.

Table 5-2 Common Liberty II Classes and Interfaces *(Continued)*

Class or Interface Name	Description
`ResourceIDMapper`	Interface Used to map between an userID and the ResourceID associated with it. Included in package `com.sun.identity.liberty.ws.common.interfaces`.

Trusted Authority

The Trust Authority component provides APIs for creating security tokens used for authentication and authorization in accordance with the ID-WSF Security Mechanisms Specification. Both WSS X509 and SAML tokens are supported.

Security Token Manager

This is the entry class for the security package `com.sun.identity.liberty.ws.security`. You can call methods in this class 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.

Table Table 5-3 summarizes the Liberty security APIs included in `com.sun.identity.liberty.ws.security`. For detailed API reference, including methods and their syntax and parameters, see the Javadocs at `/opt/SUNWam/docs/am_public_javadocs.jar`.

Table 5-3 Security APIs

Class Name	Description
`SecurityTokenProvider`	A provider interface for managing WSS security 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
`SecurityAssertion`	Provides an extension to Assertion class to support ID-WSF ResourceAccessStatement and SessionContextStatement
`SecurityTokenManager`	A final class that provides interfaces to manage Web Service Security (WSS) Tokens.
`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.

Table 5-3 Security APIs *(Continued)*

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`	.

SOAP Binding

The SOAP Binding component is designed to be a generic transport layer for handling SOAP messages.

Table 5-4 Summarizes the APIs for SOAP binding. For detailed API reference, including methods and their syntax and parameters, see the Javadocs at `/opt/SUNWam/docs/am_public_javadocs.jar`.

Table 5-4 APIs for SOAP binding.

Class Name	Description
`Message`	Used by web service client and server to construct a request or response.
`CorrelationHeader`	Represents the Correlation element defined in the SOAP binding schema.
`ConsentHeader`	Represents the Consent element defined in the SOAP binding schema.
`UsageDirectiveHeader`	This is the default constructor.
`Client`	Provides web service clients with a method to send requests using a SOAP connection to web service servers.
`SOAPReceiver`	Defines a SOAP Receiver which supports SOAP over HTTP binding.
`RequestHandler`	This interface must be implemented by web service in order to receive requests from a client.

A client-side API is provided so that any web server client can talk to the server's SOAP end point. A server-side hook, the `RequestHandler` interface, is provided so that any web service provider can easily be plugged into the Liberty Identity web service framework.

Plugin a new Web Service Provider

1. The web service provider needs to implement the `RequestHandler` interface, which will be called to do service specific processing.

Code Example 5-1 New Web Service Provider

```
/**
* The RequestHandler interface needs to be implemented
* by web services in order to receive request from client.
*/
public interface RequestHandler {
/**
* Generates a response according to the request
* @param  request request message object
* @return         response message
*/
    public Message
    processRequest(
       Message request
   )
}
```

2. The web service provider need to register the RequestHandler in the SOAP service together with a second level URI. SOAP service registers the top level servlet mapping URI in web.xml: <servlet>

```
 <servlet-name>WSSOAPReceiver</servlet-name>

<servlet-class>com.sun.identity.liberty.ws.soapbinding.SOAPReceiver</servl
et-class>
</servlet>
<servlet-mapping>
        <servlet-name>WSSOAPReceiver</servlet-name>
        <url-pattern>/Liberty/*</url-pattern>
</servlet-mapping>
```

For example, a personal profile service provider could register `idpp` URI with the SOAP service. Registration is done through Identity Server administration console by adding the key / handler class mapping to the SOAP service Request Handler List field:

`key=idpp|class=com.sun.identity.liberty.ws.idpp.PPRequestHandler`

If the SOAP service receives any requests to the `/Liberty/idpp` URI, the registered ID-SIS-PP `RequestHandler` will be invoked to process the request, and return will be sent as response to the requester.

Authorization

When a web service consumer contacts a web service provider, the consumer may convey both sender identity and invocation identity. The web service provider must make an authorization decision based on one or both identities. Identity Server depends on the policy framework to perform access evaluation.

Creating an SSO Token

Identity Server framework requires an SSO token in order to perform policy evaluation. The SSO token is created in the SOAP layer. Any web service provider can use the SSO token to perform policy evaluation.

SOAP creates the SSO token after successful peer and/or message authentication. The subject of the certification used in peer or message authentication, or in both, will be set as the subject of the SSO token. An SPI interface is defined to create an SSO token based on the SOAP message and/or HTTP servlet request.The Sender Identity, Authentication Mechanism, and SOAP Message properties will be set in the SSO token.

Creating a Policy

You must create a policy for the identity service. See the Identity Server Administration Guide for detailed steps for creating a policy. Specify the policy subject, roles, and so forth that you want to use for authorization to be enforced.

Table 5-5 summarizes the Security Token APIs included in com.sun.identity.liberty.ws.common.wsse. For detailed API reference, including methods and their syntax and parameters, see the Javadocs at `/opt/SUNWam/docs/am_public_javadocs.jar`.

Table 5-5 Binary Security Token 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`	.

Discovery Service

A Discovery Service describes and discovers identity services. By default, a Discovery service is implemented as one of the identity web services in Identity Server. Discovery Service provides four interfaces:

- Authorizer
- DefaultDiscoAuthorizer
- ResourceIDMapper
- DiscoEntryHandler

Authorizer

The class `com.sun.identity.liberty.ws.interfaces.Authorizer` is an interface to enable Identity service to check authorization of a Web Service Client.

DefaultDiscoAuthorizer

The `com.sun.identity.liberty.ws.disco.plugins.DefaultDiscoAuthorizer` is the Discovery Service default implementation. This implementation uses the policy service defined in `sunIdentityServerDiscoveryService`. In this policy definition, a policy resource uses the form:

ServiceType + *RESOURCE_SEPERATOR* + *ProviderID*.

Example:

`urn:liberty:id-sis-pp:2003-08;http://example.com`.

Policy credentials are `SSOToken for Authentication Users` or `Web Service Clients`. The actions are `DiscoConstants.ACTION_LOOKUP` or `DiscoConstants.ACTION_UPDATE`.

You can write a policy plugin to process and use the information passed in through the environment variable. Or you can define a complete new policy service and implement your own Authorizer plugin.

ResourceIDMapper

The class `com.sun.identity.liberty.ws.interfaces.ResourceIDMapper` is an interface that is used to map a `userID` to the `ResourceID` associated with it.

A different implementation of the interface may be developed by different service provider. The implementation classes should be given to the provider that hosts Discovery Service. The mapping between the `providerID` and the implementation class can be configured through the "Class for ResourceID Mapper Plugin" field in Discovery Service.

Discovery Service provides two implementations for this interface. `com.sun.identity.liberty.ws.disco.plugins.Default64ResourceIDMapper` assumes the format of ResourceID is:

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

In this example, `com.sun.identity.liberty.ws.disco.plugins.DefaultHexResourceIDMapper` assumes the format of ResourceID is:

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

DiscoEntryHandler

The class `com.sun.identity.liberty.ws.disco.plugins.DiscoEntryHandler` is an interface that is used to get and set `DiscoEntries` for a user. A default implementation is provided for this Discovery Service. If you want to handle `DiscoEntry` differently, implement this interface and set the implementing class to `DiscoEntryHandler Plugins Class` field in Discovery Service.

Discovery Service provides three implementations for this interface.

UserDiscoEntryHandler. This implementation gets or modifies discovery entries stored at the user's entry in the attribute named `sunIdentityServerDiscoEntries`. The resource offering is saved in a user entry. `UserDiscoEntryHandler` is used in business-to-consumer scenarios such as the Personal Profile service.

DynamicDiscoEntryHandler. This implementation gets discovery entries stored at the dynamic template in attribute named `sunIdentityServerDynamicDiscoEntries`. Modification method is not supported and always returns false. The resource offering is saved in an organization or a role. `DynamicDiscoEntryHandler` is used in business-to-business scenarios such as the Employee Profile service.

UserDynamicDiscoEntryHandler. This implementation gets a union of the discovery entries stored at the user entry in attribute named `sunIdentityServerDiscoEntries` and entries stored at the dynamic template in attribute named `sunIdentityServerDynamicDiscoEntries`. It modifies discovery entries stored at the user entry in the attribute named `sunIdentityServerDiscoEntries`. `UserDynamicDiscoEntryHandler` is used in both business-to-consumer and business-to-business scenarios.

Client APIs

Table 5-6 summarizes the Discovery Service client APIs included in `com.sun.identity.liberty.ws.disco`. For detailed API reference, including methods and their syntax and parameters, see the Javadocs at `/opt/SUNWam/docs/am_public_javadocs.jar`.

Table 5-6 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.
`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.
`DiscoEntryHandler`	An interface used to get and set DiscoEntries for a user. Contained in the com.sun.identity.liberty.ws.disco.plugins package.

Data Services Template

The Data Services Template is designed to be a base layer that can be extended by any data services instance. An example of a data service a personal profile service such as a 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 your colleague's identity. The information may include the individual's office location and phone number, as well as other data such as his job title and department name.

The Data Services client APIs for the template provide the building blocks for implementing a data service on top of the Identity Services framework. The template 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 data identity services must be built on top of the template. The Data Service Template provides the data model and the message interfaces for all data services.

Client APIs

Table 5-7 summarizes the Data Services client APIs included in com.sun.identity.liberty.ws.dst. For detailed API reference, including methods and their syntax and parameters, see the Javadocs at `/opt/SUNWam/docs/am_public_javadocs.jar`.

Table 5-7 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.
`DSTException`	Represents an error while processing Data Service Templates query or modify requests.

Personal Profile Service

A Personal Profile service (ID-PP) is a data-oriented identity web service hosted by an attribute provider for web service clients. It is designed to make public the user personal profiles in the World Wide Web. Examples of personal profile services used in a company are the corporate calendar or phone book which provide information associated with an employee's individual identity.

Identity Server provides a framework for developing identity web services, and also provides a built-in personal profile service that you can deploy.

Before the Web Service Client posts a query or a modify request, the Personal Profile service for a specific resource must be registered with Discovery Service. This is done by updating a resource offering for a specific resource. The invocation of the personal profile starts when a web services client posts a query or a modify request to the Personal Profile service on behalf of the user.

How It Works

1. A web services consumer posts either a query or a modify request to the Personal Profile service by keeping appropriate credentials based on the security profile that it is using to communicate.
2. The client's `SOAPRequest` is received by the `SOAPReceiver` provided by the SOAP binding framework. The SOAP framework authenticates the web services client and invokes the corresponding service.
3. The Personal Profile service handler parses the request and evaluates the authorization for each item in the request. Policy enforcement is done by making use of Identity Server Policy framework. The processing of each request is subject to the processing rules specified by the Liberty Data Template specification.
4. The Personal Profile service responds to query and modify requests. For query requests, the 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 service xml definition, 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. For modify requests, it parses the Modifiable Select expression and updates the new data from the new data node in the request.
5. Finally, the Personal Profile service builds a service response and adds credentials (if they are required), then sends it back to the web services client.

Notes on Customizing the Personal Profile Service

This Personal Profile service is completely customizable and provides multiple levels of customization that you can implement to meet your company’s needs. These include:

- Attribute Mapping
- Authorization
- Containers
- Extensions
- Rewriting the whole service

Attribute Mapping

Each Personal Profile attribute defined by the Liberty Personal Profile service has a one-to-one Identity Server Personal Profile service attribute. However, we do not need to use the same attributes, instead, we could specify attribute mappings with any other user attributes. These attribute mappings are defined as a global attribute in the personal profile service xml definition.

In the following example, Liberty `informalName` is mapped to a user's `uid` which is defined by `IDService`:

```
<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>
```

Authorization

The authorization component is a plug-in to the Personal Profile service and is defined by the service as a global attribute. The plug-in implementation must implement the `Authorizer` interface as defined by API.

The following the default plug-in implementation:

```
<AttributeSchema name="sunIdentityServerPPDSAttributeMapList"
                      type="list"
                      syntax="string"
                      i18nKey="p108">
                      <DefaultValues>
                         <Value>CN=sunIdentityServerPPCommonNameCN</Value>
                         <Value>FN=sunIdentityServerPPCommonNameFN</Value>
```

```
<AttributeSchema name="sunIdentityServerPPDSAttributeMapList"
                          <Value>MN=sunIdentityServerPPCommonNameMN</Value>
                          <Value>SN=sunIdentityServerPPCommonNameSN</Value>
                          <Value>InformalName=uid</Value>
              </AttributeSchema>
```

The default authorization plug-in uses the Identity Server policy framework and defines four different 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:

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

Here, the subjects can be defined as web services clients.

The policy enforcement can be turned off globally by a boolean flag defined by the service `xml` file.

Containers

The Liberty Personal Profile specification defines all the Personal Profile attributes as container leaf attributes so that they can be easily referencable. Identity Server defines each Personal Profile container as a pluggable entity. The service can also limit the number of personal profile containers that it would like to support by defining them as a global service attribute value.

Each container implements a container interface `IDPPContainer`. For example, `CommonName` is a Personal Profile container. The container is defined as follows.

```
public class IDPPCommonName implements IDPPContainer {
       //////////////
}
```

Note that container implementations are not configurable through Service Configuration.

Extensions

The Liberty Personal Profile service allows you to specify extension attributes that are not defined by the Liberty specification. These extensions can be specified for any container or at the container leaf level. The Identity Server Personal Profile service, however, allows you to specify only the extension attributes at the container-level extension element. For example, all the extensions should be defined as follows:

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

A typical extension query expression for an extension attribute is as follows:

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

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

Rewriting the whole service

Personal Profile Service is completely replacable by using the existing web services framework. Each web service needs to register with the SOAP service by specifying a handler. The handler extends from the SOAP's request handler.

The default Personal Profile Service is registered as follows:

```
<AttributeSchema name="RequestHandlerList"
                     type="list"
                     syntax="string"
                     uitype="name_value_list"
                     i18nKey="a101">
                     <DefaultValues>

<Value>key=disco|class=com.sun.identity.liberty.ws.disco
.DiscoveryService</Value>

<Value>key=idpp|class=com.sun.identity.liberty.ws.idpp.P
PRequestHandler</Value>
                     </DefaultValues>
                </AttributeSchema>
```

Interaction Service

An identity service provider sometimes needs to interact with the owner of the exposed resources to get the owner's consent or to get additional data. Identity Server provides a framework to support interactions between web service providers and resource owners. The framework is based upon the Liberty ID-WSF Interaction Service.

The Interaction Service specifies two different mechanisms to facilitate interactions between an identity service provider and a resource owner:

- The identity service provider and web service consumer can cooperate to redirect the resource owner to the identity service provider and back to the web service consumer.
- Interaction Service can be hosted as a web service. This service can be offered by a trusted web service consumer as well as by a dedicated provider that has a reliable means of communication with the resource owner. The identity service provider would interact with this interaction service by exchanging SOAP messages.

Interaction Service provides interfaces:

com.sun.liberty.ws.interaction.wspredirecthandler. points to the URL at which `WSPRedirectHandler` servlet is deployed.

com.sun.liberty.ws.interaction.wspstylesheet. points to the URL at which the XSLT sheet used to format `InteractionQuery` is available.

Table 5-8 summarizes the Interaction Service APIs. For detailed API reference, including methods and their syntax and parameters, see the Javadocs at `/opt/SUNWam/docs/am_public_javadocs.jar`.

Table 5-8 Interaction Service APIs

Class Name	Description
`InteractionManager`	This class provides the interface and implementation for supporting resource owner interaction.
`InteractionUtils`	Provides some utility methods that work with objects related to interaction
`JAXBObjectFactory`	

Metadata Specifications

Due to changes in Liberty Metadata specification, Liberty Service Management (SM) Configuration schema in Identity Server 6.2 is no longer compatible with that in Identity Server 6.1. SM versioning will be used to support coexistence of Identity Server 6.1 and 6.2 running against same Sun Java System directory Server. Metadata publication via queries through the DNS will not be supported in this release.

External component dependency

When upgrading from Identity Server 6.1 to 6.2, meta data migration is required to make it 6.2 compliant.

PAOS

Simple Object Access Protocol (SOAP) is a lightweight protocol for the exchange of information in a decentralized, distributed environment. SOAP enables the exchange of messages using a variety of underlying protocols. PAOS is another name for the implementation of the Liberty Reverse HTTP Binding for SOAP Specification. The use of PAOS makes possible the exchange of information between user agent hosted services and remote servers.

In a typical forward SOAP binding, an HTTP client exposes a service via a client request and a server response. For example, a cell phone user (the client) may contact his phone service provider (the service) in order to retrieve stocks quotes and weather information. The service verifies the user's identity, and responds with the requested information.

In a reverse HTTP SOAP binding, the Service Provider server plays the client role, and the client plays the server role. Technically, the initial SOAP request from the server is bound to a server HTTP response. Then the subsequent response is bound to a client request. This is why Reversed HTTP Binding for SOAP is also known as PAOS (or “SOAP” spelled backwards).

PAOS APIs

Table 5-9 summarizes the PAOS APIs included in com.sun.identity.liberty.ws.pao. Note that these APIs are used by PAOS clients which are on the HTTP server side. For detailed API reference, including methods and their syntax and parameters, see the Javadocs at `/opt/SUNWam/docs/am_public_javadocs.jar`.

Table 5-9 Summary of PAOS APIs in com.sun.identity.liberty.ws.paos

Class Name	Description
`PAOSHeader`	The PAOSHeader class is used by a web application on HTTP server side to parse a PAOS header in an HTTP request from the user agent side.
`PAOSRequest`	The PAOSRequest class is used by a web application on 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 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 from `PAOSResponse` to provide correlation if needed by API users.Code Example 5-2 illustrates how to use implement a PAOS client.

Code Example 5-2 Sample PAOSClientServlet

```
public class PAOSClientServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

            // use PAOSHeader class to parse the PAOS header
            // to get at least service information
            PAOSHeader paosHeader = null;
            try {
                paosHeader = new PAOSHeader(req);
            } catch (PAOSException pe1) {
                ...
            }
```

```
            HashMap servicesAndOptions = paosHeader.getServicesAndOptions();

            Set services = servicesAndOptions.keySet();

            // construct PAOSRequest instance
            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) {
                ...
            }

            // send PAOS request
            paosReq.send(res, true);
    }

    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

            // construct a PAOSResponse instance from the HTTP request
            PAOSResponse paosRes = null;
            try {
                paosRes = new PAOSResponse(req);
            } catch (PAOSException pe) {
                ...
            }

            // get the data from the PAOSResponse
            String dataStr = null;
            try {
                dataStr = paosRes.getPPResponseStr();
            } catch (PAOSException paose) {
                ...
            }
            ...
    }
}
```

Glossary

Refer to the Java Enterprise System glossary for a complete list of terms that are used in this documentation set.

http://docs.sun.com/source/816-6873/index.html

Index

SYMBOLS

A

B

C

D

E

F

I

K

L

M

N

O

P

R

S

T

U

W

X