**For Review Purposes Only**

# Sun Federated Access Manager 8.0 Technical Overview

Beta

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

Part No: 820–3740–10
September 2008

Copyright 2008 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

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

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

This distribution may include materials developed by third parties.

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

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

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

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

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

---

Copyright 2008 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

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

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

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

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

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

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

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

080730@20490

# List of Remarks

**REMARK 1–1** **Reviewer** more details to come from Indira per review ........ 24

**REMARK 2–1** **Reviewer** Please confirm these. I have gotten different comments from different reviewers. ........ 58

**REMARK 6–1** **Reviewer** There are 13 bullets in graphic and nine steps? Huh????? What should be changed? From email with subject: Auth/SSO process question not answered. check against new graphic ........ 93

# Contents

**Preface** ....................11

**Part I** **An Overview of Federated Access Manager** .................... 17

**1** **Introducing Federated Access Manager** .................... 19
What is Federated Access Manager? .................... 19
What Does Federated Access Manager Do? .................... 20
What Are the Functions of Federated Access Manager? .................... 21
  Access Control .................... 21
  Federation Management .................... 22
  Web Services Security .................... 22
  Identity Web Services .................... 23
What Else Does Federated Access Manager Offer? .................... 23

**2** **Examining Federated Access Manager** .................... 27
Federated Access Manager Client/Server Architecture .................... 27
How Federated Access Manager Works .................... 29
Core Services .................... 31
  Authentication Service .................... 32
  Policy Service .................... 35
  Session Service .................... 37
  Logging Service .................... 40
  Identity Repository Service .................... 42
  Federation Services .................... 44
  Web Services Stack .................... 46
  Web Services Security .................... 47
  Identity Web Services .................... 48

Global Services ..... 50
Realms ..... 51
Additional Components ..... 54
  Data and Data Stores ..... 54
  The `bootstrap` File ..... 60
  Policy Agents ..... 61
  Authentication Agents ..... 62
  Federated Access Manager Tools ..... 62
  Client SDK ..... 63
  Service Provider Interfaces for Plug-ins ..... 63

**3 Simplifying Federated Access Manager** ..... 65
Installation and Configuration ..... 65
Embedded Configuration Data ..... 67
Centralized Agent Configuration ..... 68
Common Tasks ..... 70
Third Party Integration ..... 71
  Sun Java System Identity Manager ..... 72
  Computer Associates SiteMinder ..... 72
  Oracle Access Manager ..... 72

**4 Deploying Federated Access Manager** ..... 73
Deployment Architecture 1 ..... 73
Deployment Architecture 2 ..... 74

**Part II Access Control Using Federated Access Manager** ..... 77

**5 User Sessions and the Session Service** ..... 79
About the Session Service ..... 79
User Sessions and Single Sign-on ..... 80
Session Data Structures and Session Token Identifiers ..... 81

**6 Models of the User Session and Single Sign-On Processes** ..... 83
Basic User Session ..... 83

Initial HTTP Request ........ 83
User Authentication ........ 85
Session Validation ........ 87
Policy Evaluation and Enforcement ........ 89
Logging the Results ........ 91
Single Sign-On Session ........ 93
Cross-Domain Single Sign-On Session ........ 95
Session Termination ........ 97
User Ends Session ........ 97
Administrator Ends Session ........ 98
Federated Access Manager Enforces Timeout Rules ........ 98
Session Quota Constraints ........ 98

**7 Authentication and the Authentication Service ........ 99**
Authentication Service Overview ........ 99
Authentication Service Features ........ 102
Client Detection ........ 102
Account Locking ........ 102
Authentication Chaining ........ 103
Fully Qualified Domain Name Mapping ........ 104
Persistent Cookies ........ 104
Session Upgrade ........ 105
JAAS Shared State ........ 105
Security ........ 105
Authentication Modules ........ 106
Authentication Types ........ 108
Configuring for Authentication ........ 110
Core Authentication Module and Realm Configuration ........ 110
Authentication Configuration Service ........ 111
Login URLs and Redirection URLs ........ 111
Authentication Graphical User Interfaces ........ 111
Authentication Service User Interface ........ 111
Distributed Authentication User Interface ........ 113
Authentication Service Programming Interfaces ........ 115

**8 Authorization and the Policy Service** 117
Authorization and Policy Service Overview 117
Policy Types 119
  Normal Policy 119
  Referral Policy 122
Realms and Access Control 122
Policy Service Programming Interfaces 123
XACML Service 123
  XACML in Federated Access Manager 124
  XACML Programming Interfaces 127

**Part III Federation Management Using Federated Access Manager** 129

**9 What is Federation?** 131
The Concept of Federation 131
  Identity Federation 131
  Provider Federation 132
The Concept of Trust 133
How Federation Works 133

**10 Federation Management with Federated Access Manager** 137
Key Federation Management Features 137
  The Fedlet 138
  Virtual Federation 138
  Multi-Federation Protocol Hub 141
The Federation Framework Architecture 142

**11 Choosing a Federation Option** 145
Federation Options 145
Using SAML 146
  About SAML v2 148
  About SAML v1.x 152
  Using SAML or Federated Access Manager CDSSO 153
Using the Liberty ID-FF 154

- Liberty ID-FF Features ... 155
- About the Liberty ID-FF Process ... 161

Using WS-Federation ... 164

**Part IV Delivering Identity Services, Web Services and Web Services Security** ... 169

**12 Delivering Identity Web Services** ... 171

About Identity Web Services ... 171

Identity Web Service Styles ... 172
- SOAP and WSDL ... 172
- REST ... 173

Identity Web Services Architecture ... 174

**13 Accessing and Securing Web Services** ... 175

About the Web Services Stack ... 175

Web Services Architecture ... 177
- Examining Browser Mode Architecture ... 179
- Desktop Mode Architecture ... 182

Implemented Services ... 184

Web Services Process ... 184

**Part V Additional Features** ... 187

**14 Logging and the Java Enterprise System Monitoring Framework** ... 189

Logging Overview ... 189
- Logging Service ... 190
- Logging Configuration ... 190
- Recorded Events ... 190

Log Files ... 191
- Log File Formats ... 191
- Error and Access Logs ... 193

Access Manager Component Logs ... 194

Additional Logging Features ... 195
- Secure Logging ... 195

Remote Logging ..... 195
Log Reading ..... 196
Java Enterprise System Monitoring Framework ..... 196

**Index** ..... 197

# Preface

Sun Federated Access Manager is an access management product that includes a set of software components to provide the authentication and authorization services needed to support enterprise applications distributed across a network or Internet environment. This book, *Sun Federated Access Manager 8.0 Technical Overview*, describes the features of Federated Access Manager, explains what it does, and illustrates how it works.

## Before You Read This Book

This book is intended for use by IT administrators and software developers who implement a web access platform using Sun servers and software. Readers of this guide should be familiar with the following:

- Web containers in which Federated Access Manager can be deployed:
  - Sun Java System Application Server
  - Sun Java System Web Server
  - BEA WebLogic
  - IBM WebSphere Application Server
- Technologies:
  - Lightweight Directory Access Protocol (LDAP)
  - Java™
  - JavaServer Pages™ (JSP)
  - HyperText Transfer Protocol (HTTP)
  - HyperText Markup Language (HTML)
  - eXtensible Markup Language (XML)
  - SOAP
  - HyperText Transfer Protocol (HTTP)
  - Liberty Alliance Project specifications

# Related Books

Related documentation is available as follows:

- "Federated Access Manager Core Documentation" on page 12
- "Adjunct Product Documentation" on page 13

## Federated Access Manager Core Documentation

The Federated Access Manager core documentation set contains the following titles:

- The *Sun Federated Access Manager 8.0 Early Access (EA) Release Notes* will be available online after the product is released. It gathers an assortment of last-minute information, including a description of what is new in this current release, known problems and limitations, installation notes, and how to report issues with the software or the documentation.
- The *Sun Federated Access Manager 8.0 Technical Overview* (this guide) provides an overview of how Federated Access Manager components work together to protect enterprise assets and web-based applications. It also explains basic concepts and terminology.
- The XXXXX provides planning and deployment solutions for Sun Java System Access Manager based on the solution life cycle
- The *Sun Federated Access Manager 8.0 Installation and Configuration Guide* provides information for installing and configuring Federated Access Manager.
- The XXXXX provides information on how to tune Access Manager and its related components for optimal performance.
- The *Sun Federated Access Manager 8.0 Administration Guide* describes administrative tasks such as *how to create a realm* and *how to configure a policy*. Most of the tasks described can be performed using the administration console as well as the `famadm` command line utilities.
- The *Sun Federated Access Manager Administration Reference* is a look-up guide containing information about the command line interfaces, configuration attributes, internal files, and error codes.
- The *Sun Federated Access Manager 8.0 Developer's Guide* offers information on how to customize Access Manager and integrate its functionality into an organization's current technical infrastructure. It also contains details about the programmatic aspects of the product and its API.
- The *Sun Federated Access Manager 8.0 C API Reference* provides summaries of data types, structures, and functions that make up the public Access Manager C APIs.
- The *Federated Access Manager 8.0 Java API Reference* provides information about the implementation of Java packages in Access Manager.

- The *Sun Java System Federated Access Manager Policy Agent 3.0 User's Guide* provides an overview of the policy functionality and the policy agents available for Federated Access Manager.

Updates to the *Release Notes* and links to modifications of the core documentation can be found on the Federated Access Manager page at docs.sun.com. Updated documents will be marked with a revision date.

## Adjunct Product Documentation

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

- Sun Java System Directory Server Enterprise Edition 6.0
- Sun Java System Web Server 7.0
- Sun Java System Application Server Enterprise Edition 8.2
- Sun Java System Web Proxy Server 4.0.4

# Searching Sun Product Documentation

Besides searching Sun product documentation from the docs.sun.com$^{SM}$ web site, you can use a search engine by typing the following syntax in the search field:

*search-term* `site:docs.sun.com`

For example, to search for "broker," type the following:

```
broker site:docs.sun.com
```

To include other Sun web sites in your search (for example, java.sun.com, www.sun.com, and developers.sun.com), use `sun.com` in place of `docs.sun.com` in the search field.

# Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (`http://www.sun.com/documentation/`)
- Support (`http://www.sun.com/support/`)
- Training (`http://www.sun.com/training/`)

# Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

---

**Note –** Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

---

# Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to `http://docs.sun.com` and click Send Comments. In the online form, provide the full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the title of this book is *Sun Federated Access Manager 8.0 Technical Overview*, and the part number is 820–3740.

# Typographic Conventions

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

**TABLE P–1** Typographic Conventions

| Typeface | Meaning | Example |
|---|---|---|
| `AaBbCc123` | The names of commands, files, and directories, and onscreen computer output | Edit your `.login` file.<br>Use `ls -a` to list all files.<br>`machine_name% you have mail.` |
| **`AaBbCc123`** | What you type, contrasted with onscreen computer output | `machine_name%` **`su`**<br>`Password:` |
| *AaBbCc123* | A placeholder to be replaced with a real name or value | The command to remove a file is `rm` *filename*. |

**TABLE P–1** Typographic Conventions *(Continued)*

| Typeface | Meaning | Example |
|---|---|---|
| *AaBbCc123* | Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online) | Read Chapter 6 in the *User's Guide*.<br>A *cache* is a copy that is stored locally.<br>Do *not* save the file. |

# Shell Prompts in Command Examples

The following table shows default system prompts and superuser prompts.

**TABLE P–2** Shell Prompts

| Shell | Prompt |
|---|---|
| C shell on UNIX and Linux systems | `machine_name%` |
| C shell superuser on UNIX and Linux systems | `machine_name#` |
| Bourne shell and Korn shell on UNIX and Linux systems | `$` |
| Bourne shell and Korn shell superuser on UNIX and Linux systems | `#` |
| Microsoft Windows command line | `C:\` |

# Symbol Conventions

The following table explains symbols that might be used in this book.

**TABLE P–3** Symbol Conventions

| Symbol | Description | Example | Meaning |
|---|---|---|---|
| `[ ]` | Contains optional arguments and command options. | `ls [-l]` | The `-l` option is not required. |
| `{ \| }` | Contains a set of choices for a required command option. | `-d {y\|n}` | The `-d` option requires that you use either the `y` argument or the `n` argument. |
| `${ }` | Indicates a variable reference. | `${com.sun.javaRoot}` | References the value of the `com.sun.javaRoot` variable. |
| - | Joins simultaneous multiple keystrokes. | Control-A | Press the Control key while you press the A key. |

**TABLE P–3** Symbol Conventions *(Continued)*

| Symbol | Description | Example | Meaning |
|---|---|---|---|
| + | Joins consecutive multiple keystrokes. | Ctrl+A+N | Press the Control key, release it, and then press the subsequent keys. |
| → | Indicates menu item selection in a graphical user interface. | File → New → Templates | From the File menu, choose New. From the New submenu, choose Templates. |

# PART I

# An Overview of Federated Access Manager

This part of the Sun Federated Access Manager Technical Overview contains introductory material concerning Federated Access Manager. It includes the following chapters:

- Chapter 1, "Introducing Federated Access Manager"
- Chapter 2, "Examining Federated Access Manager"
- Chapter 3, "Simplifying Federated Access Manager"

◆ ◆ ◆

## CHAPTER 1

# Introducing Federated Access Manager

Sun Java™ System Federated Access Manager integrates authentication and authorization services, single sign-on (SSO), and open, standards-based federation protocols (including the Liberty Alliance Project specifications, WS-Federation and Security Assertion Markup Language [SAML]) to provide a comprehensive solution for protecting network resources by preventing unauthorized access to web services, applications and web content, and securing identity data. This introductory chapter contains a high-level description of Federated Access Manager and what it does. It contains the following sections:

- "What is Federated Access Manager?" on page 19
- "What Does Federated Access Manager Do?" on page 20
- "What Are the Functions of Federated Access Manager?" on page 21
- "What Else Does Federated Access Manager Offer?" on page 23

## What is Federated Access Manager?

Sun Federated Access Manager is a single product that combines the features of Sun Java System Access Manager, Sun Java System Federation Manager, and the Sun Java System SAML v2 Plug-in for Federation Services; additionally, it is enhanced with new functionality developed specifically for this release. Federated Access Manager provides access management by allowing the implementation of authentication, policy-based authorization, federation, SSO, and web services security from a single, unified framework. The core application is delivered as a simple web archive (WAR) that can be easily deployed in a supported web container.

---

**Note –** Federated Access Manager is Sun Microsystems' commercial distribution of the open source code available at OpenSSO.

---

To assist the core application, policy agents, the Client SDK, and (possibly) other disparate pieces must be installed remotely and be able to communicate with the Federated Access

Manager server. See "What Does Federated Access Manager Do?" on page 20 for a high-level explanation of the deployment architecture and Chapter 2, "Examining Federated Access Manager," for more specific information.

# What Does Federated Access Manager Do?

The following types of interactions occur daily in a corporate environment.

- An employee looks up a colleague's phone number in the corporate phone directory.
- A manager retrieves employee salary histories to determine an individual's merit raise.
- An administrative assistant adds a new hire to the corporate database, triggering the company's health insurance provider to add the new hire to its enrollment.
- An engineer sends an internal URL for a specification document to another engineer who works for a partner company.
- A customer logs into a company's web site and looks for a product in their online catalog.
- A vendor submits an invoice to the company's accounting department.
- A corporate human resources administrator accesses an outsourced benefits application.

For each of these transactions, the company must determine who is allowed to view the information or use the application. Some information such as product descriptions and advertising can be made available to everyone in a public online catalog. Other information such as accounting and human resources data must be restricted to employees only. And other sensitive information such as pricing models and employee insurance plans is appropriate to share only with partners, suppliers, and employees. This need for access determination is met by Sun Federated Access Manager, an access management product with authentication, authorization, and single sign-on (SSO) services provided out of the box.

When a user or an external application requests access to content stored on a company's server, a *policy agent* (available in a separate download and installed on the same machine as the resource you want to protect) intercepts the request and directs it to Federated Access Manager which, in turn, requests credentials (such as a username and password in the case of a user) for authentication. If the credentials returned match those stored in the appropriate identity data store, Federated Access Manager determines that the user is authentic. Following authentication, access to the requested content is determined by the policy agent which evaluates the policies associated with the authenticated identity. Policies are created using Federated Access Manager and identify which identities are allowed to access a particular resource, specifying the conditions under which this authorization is valid. Based upon the results of the policy evaluation, the policy agent either grants or denies the user access. Figure 1–1 illustrates a high-level deployment architecture of Federated Access Manager.

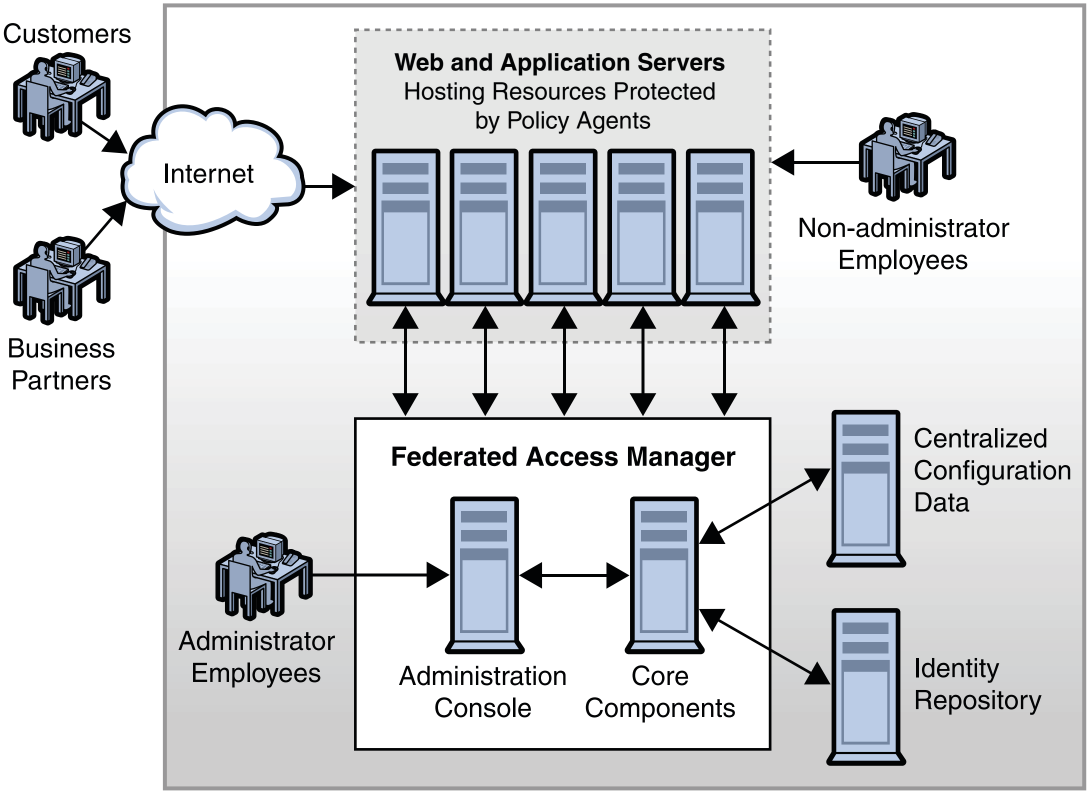

Customers
Internet
Business Partners
Web and Application Servers Hosting Resources Protected by Policy Agents
Non-administrator Employees
Federated Access Manager
Centralized Configuration Data
Administrator Employees
Administration Console
Core Components
Identity Repository

FIGURE 1–1 High-level Deployment Architecture of Federated Access Manager

# What Are the Functions of Federated Access Manager?

The following sections contain an overview of the functions of Federated Access Manager.

- "Access Control" on page 21
- "Federation Management" on page 22
- "Web Services Security" on page 22
- "Identity Web Services" on page 23

## Access Control

Federated Access Manager manages authorized access to network services and resources. By implementing authentication and authorization, Federated Access Manager (along with an installed policy agent) ensures that access to protected resources is restricted to authorized users. In a nutshell, a policy agent intercepts a request for access to a resource and communicates with Federated Access Manager to authenticate the requestor. If the user is successfully authenticated, the policy agent then evaluates the policies associated with the requested resource and the user to determine if the authenticated user is authorized to access the resource. If the user is authorized, the policy agent allows access to the resource, also providing identity data to the resource to personalize the interaction. For more information on access control, see "Core Services" on page 31 and Part II.

## Federation Management

With the introduction of federation protocols into the process of access management, identity information and entitlements can be communicated across security domains, spanning multiple trusted partners. By configuring a *circle of trust* and defining applications and services as *providers* in the circle (either identity providers or service providers), users can opt to associate, connect or bind the various identities they have configured locally for these providers. The linked local identities are federated and allow the user to log in to one identity provider site and click through to an affiliated service provider site without having to reauthenticate; in effect, single sign-on (SSO). Federated Access Manager supports several open federation technologies including the Security Access Markup Language (SAML) versions 1 and 2, WS-Federation, and the Liberty Alliance Project Identity Federation Framework (Liberty ID-FF), therefore encouraging an interoperable infrastructure among providers. For more information on federation management, see "Core Services" on page 31 and Part III.

## Web Services Security

A *web service* is a component service or application that exposes some type of business or infrastructure functionality through a language-neutral and platform-independent, network interface; enterprises might use this web service to build larger service-oriented architectures. In particular, the service defines its interface (for example, the format of the message being exchanged) using the Web Services Description Language (WSDL), and communicates using SOAP and eXtensible Markup Language (XML) messages. The web service client (WSC) communicates with the web service provider (WSP) through an intermediary — usually a firewall or load balancer.

Although web services enable open, flexible, and adaptive interfaces, their openness creates security risks. Without proper security protections, a web service can expose vulnerabilities that might have dire consequences. Hence, ensuring the integrity, confidentiality and security of web services through the application of a comprehensive security model is critical for both enterprises and consumers. A successful security model associates identity data with the web services and creates secure service-to-service interactions. The security model adopted by Federated Access Manager identifies the user and preserves that identity through multiple interactions, maintains privacy and data integrity, uses existing technologies, and logs the interactions. In Federated Access Manager, the following web service security standards are implemented:

- Liberty Alliance Project Identity Web Services Framework (Liberty ID-WSF)

- WS-I Basic Security Profile
- WS-Trust (from which the Security Token Service was developed)

For more information on Federated Access Manager web services and web services security, see "Core Services" on page 31 and Part IV.

## Identity Web Services

For some time, Federated Access Manager has provided client interfaces for access to core features and functionality. These interfaces are used by policy agents and custom applications developed by customers. With this release, Federated Access Manager now exposes certain functions as simple identity web services allowing developers to easily invoke them when developing their applications using one of the supported integrated development environment (IDE) products. (The IDE generates the stub code that wraps a call to the web service.) Identity Web Services are available using:

- SOAP and Web Services Description Language (WSDL)
- Representational State Transfer (REST)

They do not require the deployment of an agent or a proxy and include the following capabilities:

- Authentication to validate user credentials.
- Authorization to permit access to protected resources.
- Provisioning for user attribute management and self registration.
- Logging to keep track of it all.

For more information on identity services, see "Identity Web Services" on page 48 and Part IV.

# What Else Does Federated Access Manager Offer?

Federated Access Manager allows for:

- **Ease of Deployment:** Federated Access Manager is delivered as a web archive (WAR) that can be easily deployed as a Java EE application in different web containers. Most configuration files and required libraries are inside the WAR to avoid the manipulation of the classpath in the web container's configuration file. The Federated Access Manager WAR is supported on:
  1. Sun Java System Web Server 7.0 — Update 3 and above
  2. Sun Java System Application Server 9.1 EE Update 2 and above (and Glassfish v2 update 2 and above)
  3. BEA WebLogic Application Server 9.2 mp2
  4. IBM WebSphere Application Server 6.1

5. Oracle Application Server 10g
6. JBoss 4.2.x
7. **[Remark 1–1 Reviewer: more details to come from Indira per review]** Tomcat 5.5.x & 6.x
8. Geronimo (supported on the Sun Solaris™ 10 Operating Environment for SPARC, x86 & x64 and the Sun Solaris 9 Operating Environment for SPARC & x86 systems only)

---

**Note –** Geronimo can install Tomcat and Jetty web containers; Federated Access Manager supports only Tomcat.

---

See the *Sun Federated Access Manager 8.0 Early Access (EA) Release Notes* for updates to this list.

- **Portability:** Federated Access Manager is supported on the following operating systems:

1. Sun Solaris 10 Operating Environment for SPARC, x86 & x64 systems
2. Sun Solaris 9 Operating Environment for SPARC & x86 systems
3. Windows Server 2003 and Windows XP (development only) operating systems
4. Red Hat Enterprise Linux 4 Server (Base)
5. Red Hat Enterprise Linux 4 Advanced Platform
6. Red Hat Enterprise Linux 5 Server (Base)
7. Red Hat Enterprise Linux 5 Advanced Platform
8. Windows 2003 Standard Server
9. Windows 2003 Enterprise Server
10. Windows 2003 Datacenter Server
11. Windows Vista
12. IBM AIX 5.3 (supported with the IBM WebSphere Application Server 6.1 container only)

See the *Sun Federated Access Manager 8.0 Early Access (EA) Release Notes* for updates to this list.

- **Open Standards:** Federated Access Manager is built using open standards and specifications as far as possible. For example, features designed for federation management and web services security are based on the Security Assertion Markup Language (SAML), the Liberty Alliance Project specifications, and the WS-Security standards.
- **Ease of Administration:** Federated Access Manager contains a web-based, graphical administration console as well as command line interfaces for configuration tasks and administrative operations. Additionally, an embedded, centralized data store allows for one place to store server and agent configuration data.
- **Security:**

1. Runtime security enables an enterprise's resources to be protected as configured and Federated Access Manager services to be accessed by authorized entities only.
2. Administration security ensures only authorized updates are made to the Federated Access Manager configuration data.
3. Deployment security implements best practices for installing Federated Access Manager on different operating systems, web containers, and so forth.

Additionally, all security actions are logged.

- **Configuration Data Store:** Federated Access Manager can write server configuration data to an embedded configuration data store. You can also point to instances of Sun Java System Directory Server 5.2 or Directory Server Enterprise Edition 6.x during configuration of Federated Access Manager for use as a configuration data store. See "Data and Data Stores" on page 54 for more information.
- **User Data Store Independence:** Federated Access Manager allows you to view and retrieve user information without making changes to an existing user database. Supported directory servers include Directory Server 5.1, 5.2 & 6.2, IBM Tivoli Directory 6.1, and Microsoft Active Directory 2003. See "Data and Data Stores" on page 54 for more information.

**Caution –** The configuration data store embedded with Federated Access Manager should only be used as a user data store for proof of concepts and deployments in development.

- **Web and Non-Web-Based Resources:** The core design of Federated Access Manager caters to SSO for both web and non-web applications.
- **Performance, Scalability and Availability:** Federated Access Manager can be scaled horizontally and vertically to handle increased workloads, and as security needs change over time. There is no single point of failure.
- **Distributed Architecture** Server and client components can be deployed across the enterprise or across domain boundaries as all application programming interfaces (API) provide remote access to Federated Access Manager based on a service-oriented architecture.
- **Flexibility and Extensibility:** Many Federated Access Manager services expose a service provider interface (SPI) allowing expansion of the framework to provide for specific deployment needs.
- **Internationalization** Federated Access Manager contains a framework for multiple language support. Customer facing messages, API, command line interfaces, and user interfaces are localized in the supported languages.

◆ ◆ ◆

## CHAPTER 2

# Examining Federated Access Manager

Federated Access Manager provides a pluggable architecture to deliver access management, secure web services, and federation capabilities. This chapter contains information on the internal architecture and features of Federated Access Manager.

- "Federated Access Manager Client/Server Architecture" on page 27
- "How Federated Access Manager Works" on page 29
- "Core Services" on page 31
- "Global Services" on page 50
- "Additional Components" on page 54

## Federated Access Manager Client/Server Architecture

Federated Access Manager is written in Java, and leverages many industry standards, including the HyperText Transfer Protocol (HTTP), the eXtensible Markup Language (XML), the Security Assertion Markup Language (SAML), and SOAP, to deliver access management, secure web services, and federation capabilities in a single deployment. It consists of client application programming interfaces (a Client Software Development Kit [SDK]), a framework of services that implement the business logic, and service provider interfaces (SPI) that are implemented by concrete classes and can be used to extend the functionality of Federated Access Manager as well as retrieve information from data stores. Figure 2–1 illustrates the client/server architecture of Federated Access Manager.

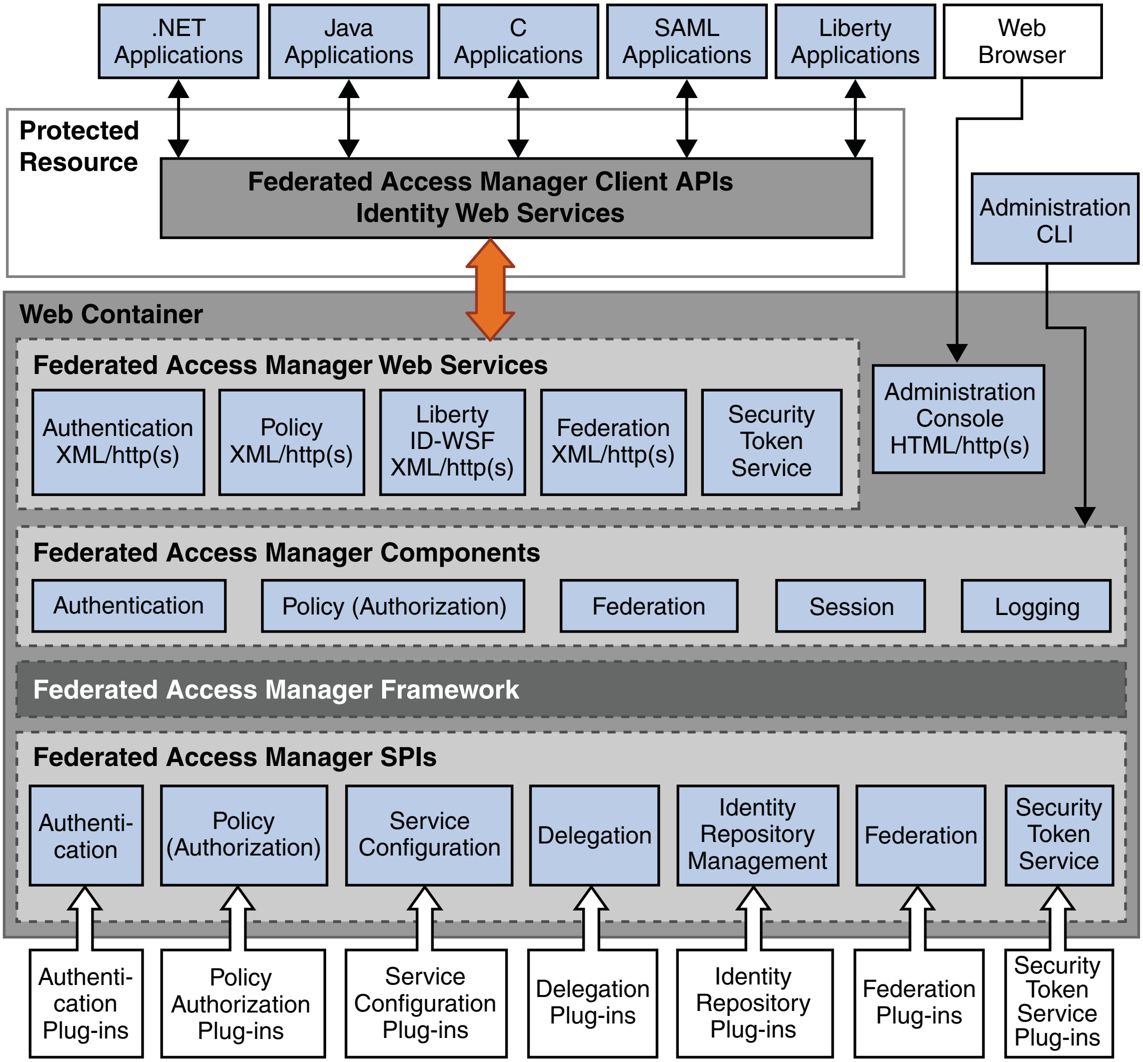

.NET Applications
Java Applications
C Applications
SAML Applications
Liberty Applications
Web Browser
Protected Resource
Federated Access Manager Client APIs
Identity Web Services
Administration CLI
Web Container
Federated Access Manager Web Services
Authentication XML/http(s)
Policy XML/http(s)
Liberty ID-WSF XML/http(s)
Federation XML/http(s)
Security Token Service
Administration Console HTML/http(s)
Federated Access Manager Components
Authentication
Policy (Authorization)
Federation
Session
Logging
Federated Access Manager Framework
Federated Access Manager SPIs
Authentication
Policy (Authorization)
Service Configuration
Delegation
Identity Repository Management
Federation
Security Token Service
Authentication Plug-ins
Policy Authorization Plug-ins
Service Configuration Plug-ins
Delegation Plug-ins
Identity Repository Plug-ins
Federation Plug-ins
Security Token Service Plug-ins

FIGURE 2–1 Client/Server Architecture of Federated Access Manager

Each component of Federated Access Manager uses its own framework to retrieve customer data from the plug-in layer and to provide data to other components. The Federated Access Manager framework integrates all of the application logic into one layer that is accessible to all components and plug-ins. The Client SDK and Identity Web Services are installed on a machine remote to the Federated Access Manager server that holds a resource to be protected; they provide remote access to the Federated Access Manager for client applications. (The policy agent, also installed on the remote machine, is basically a client written using the Client SDK and Identity Web Services.) Applications on the remote machine access Federated Access

Manager using the Client SDK. Custom plug-in modules are installed on the machine local to Federated Access Manager and interact with the Federated Access Manager SPI to retrieve required information from the appropriate data store and deliver it to the plug-ins and, in turn, the Federated Access Manager framework for processing.

# How Federated Access Manager Works

To gain access to a protected resource, the requestor needs to be authenticated and have the authorization to access the resource. When someone (using a browser) sends an HTTP request for access to a protected resource, a policy agent (separately downloaded and installed on the same machine as the resource you want to protect) intercepts the request and examines it. If no valid Federated Access Manager *session token* (to provide proof of authentication) is found, the policy agent contacts the server which then invokes the authentication and authorization processes. Figure 2–2 illustrates one way in which the policy agents can be situated to protect an enterprise's servers by directing HTTP requests to a centralized Federated Access Manager for processing.

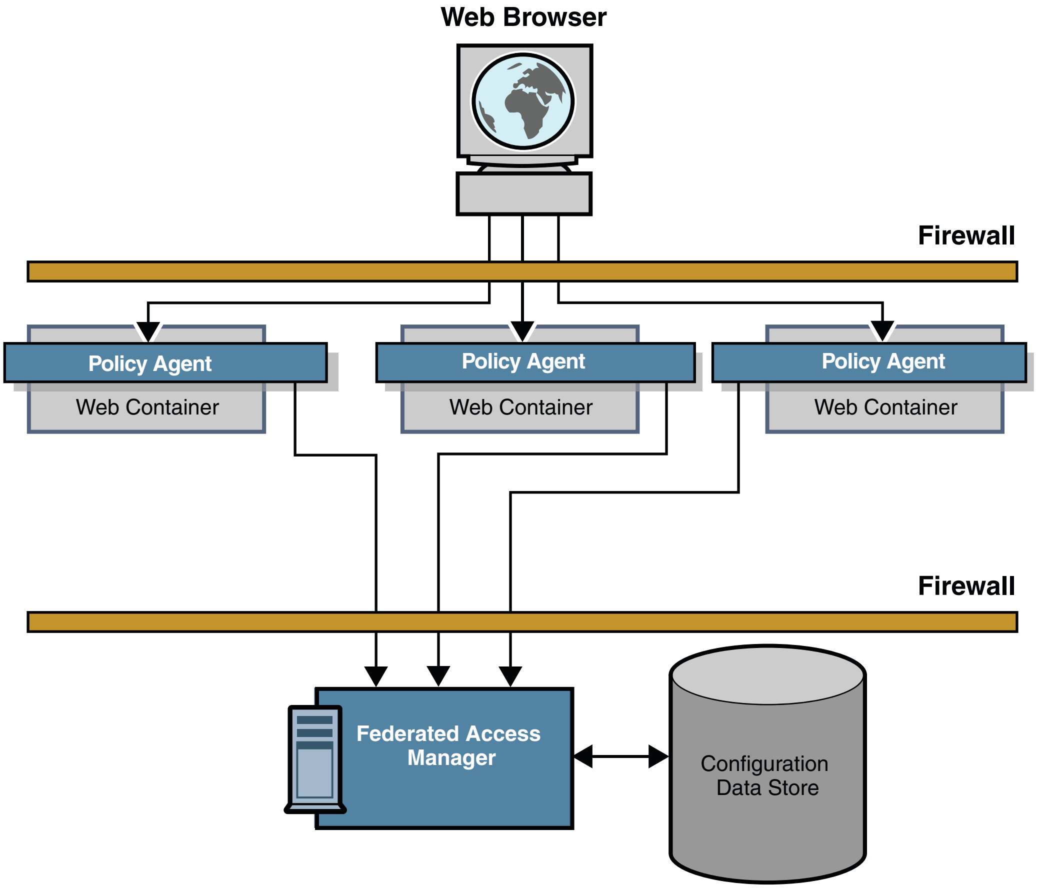

Web Browser
Firewall
Policy Agent
Web Container
Policy Agent
Web Container
Policy Agent
Web Container
Firewall
Federated Access Manager
Configuration Data Store

FIGURE 2–2 Basic Federated Access Manager Deployment

Federated Access Manager integrates core features such as access control through authentication and authorization processes, and federation. These functions can be configured using the administration console or the `famadm` command line utility. Figure 2–3 is a high-level illustration of the interactions that occur between parties (including the policy agent, browser, and protected application) during authentication and authorization in a Federated Access Manager deployment.

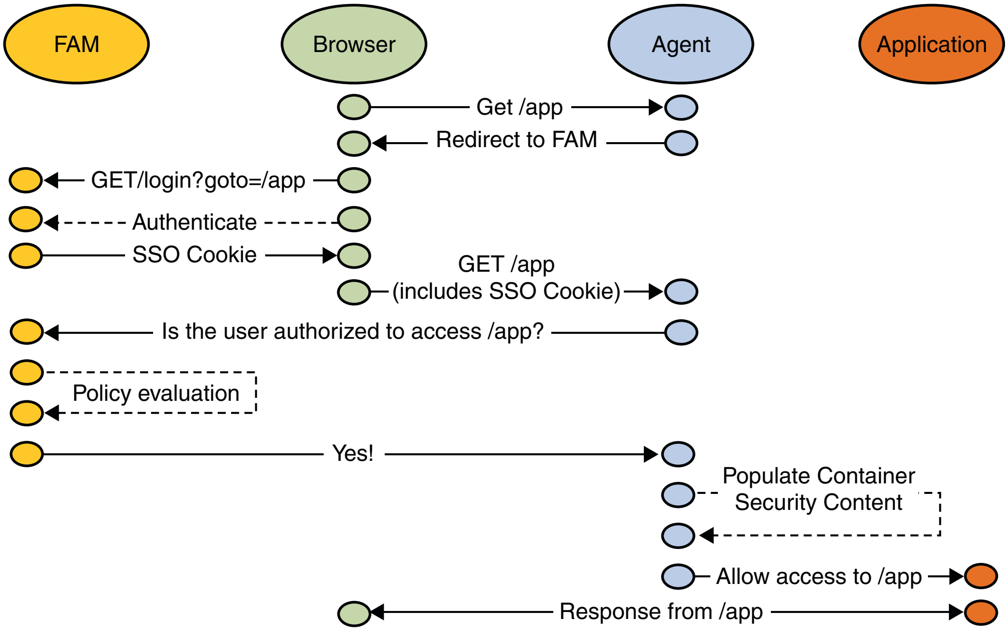

FAM
Browser
Agent
Application
Get /app
Redirect to FAM
GET/login?goto=/app
Authenticate
SSO Cookie
GET /app
(includes SSO Cookie)
Is the user authorized to access /app?
Policy evaluation
Yes!
Populate Container
Security Content
Allow access to /app
Response from /app

**FIGURE 2–3** Federated Access Manager Authentication and Authorization Interactions

For more information, see Part II.

# Core Services

Services developed for Federated Access Manager generally contain both a server component and a client component. The server component is a simple Java servlet developed to receive XML requests and return XML responses. (The deployment descriptor `web.xml` defines the servlet name and description, the servlet class, initialization parameters, mappings, and other startup information.) The client component is provided as Java application programming interfaces (API), and in some cases C API, that allow remote applications and other Federated Access Manager services to communicate with and consume the particular functionality.

Each core service uses its own framework to retrieve customer and service data and to provide it to other Federated Access Manager services. The Federated Access Manager framework integrates all of these service frameworks to form a layer that is accessible to all product components and plug-ins. The following sections contain information on the Federated Access Manager core services.

- "Authentication Service" on page 32
- "Policy Service" on page 35
- "Session Service" on page 37
- "Logging Service" on page 40

- “Identity Repository Service” on page 42
- “Federation Services” on page 44
- “Web Services Stack” on page 46
- “Web Services Security” on page 47
- “Identity Web Services” on page 48

---

**Note –** Many services also provide a public SPI that allows the service to be extended. See the *Sun Federated Access Manager 8.0 Developer's Guide*, the *Sun Federated Access Manager 8.0 C API Reference*, and the *Federated Access Manager 8.0 Java API Reference* for information.

---

# Authentication Service

The Authentication Service provides the functionality to request user credentials and validate them against a specified authentication data store. Upon successful authentication, it creates a session data structure for the user that can be validated across all web applications participating in an SSO environment. Several authentication modules are supplied with Federated Access Manager, and new modules can be plugged-in using the Java Authentication and Authorization Service (JAAS) SPI.

---

**Note –** The Authentication Service is based on the JAAS specification, a set of API that enables services to authenticate and enforce access controls upon users. See the Java Authentication and Authorization Service Reference Guide for more information.

---

Components of the Authentication Service include:

- The Distributed Authentication User Interface allows the Authentication Service user interface to be deployed separately from Federated Access Manager, if desired. By deploying this authentication proxy in the DMZ and using the authentication interfaces provided in the Client SDK to pass user credentials back and forth, you can protect Federated Access Manager data (for example, the login URL information and hence the host information). JavaServer Pages (JSP) represent the interface displayed to users for authentication and are completely customizable.
- The Core Authentication Service executes common processes across all authentication modules. Key responsibilities of this service include identification of the appropriate plan to authenticate the user (identify the authentication module, load the appropriate JSP) and creation of the appropriate session for the authenticated user.
- The Authentication API are *remoteable* interfaces that don't need to reside on the same machine as the Federated Access Manager server. This allows remote clients to access the Authentication Service. remote-auth.dtd defines the structure for the XML communications that will be used by the Authentication Service, providing definitions to initiate the process, collect credentials and perform authentication.

- A number of authentication modules are installed and configured (including, but not limited to, LDAP, RADIUS, Windows Desktop, Certificate, and Active Directory). A configured authentication level for each module is globally defined. Mechanisms are also provided to upgrade a user's session after authenticating the user to an additional authentication module that satisfies the authentication level of the resource. New modules can be plugged-in using the JAAS SPI.

The Authentication Service interacts with both the database that stores user credentials (authentication data store) to validate the user, and with the Identity Repository Service plug-ins to retrieve user profile attributes. When the Authentication Service determines that a user’s credentials are genuine, a valid user session token is issued, and the user is said to be *authenticated*. Figure 2–4 illustrates how the local and remote authentication components interact within a Federated Access Manager deployment.

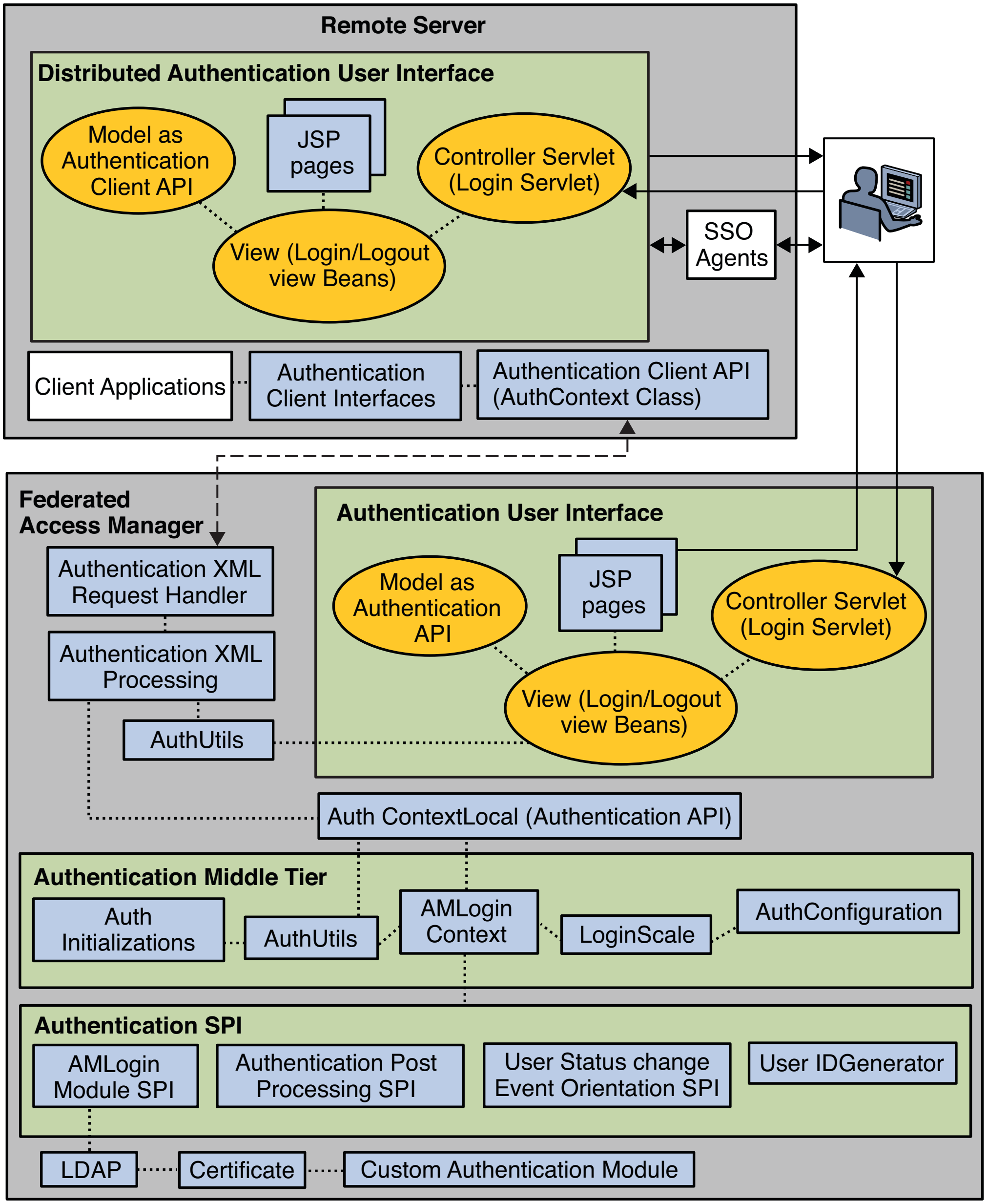

Remote Server
Distributed Authentication User Interface
Model as Authentication Client API
JSP pages
Controller Servlet (Login Servlet)
View (Login/Logout view Beans)
SSO Agents
Client Applications
Authentication Client Interfaces
Authentication Client API (AuthContext Class)
Federated Access Manager
Authentication User Interface
Authentication XML Request Handler
Authentication XML Processing
AuthUtils
Model as Authentication API
JSP pages
Controller Servlet (Login Servlet)
View (Login/Logout view Beans)
Auth ContextLocal (Authentication API)
Authentication Middle Tier
Auth Initializations
AuthUtils
AMLogin Context
LoginScale
AuthConfiguration
Authentication SPI
AMLogin Module SPI
Authentication Post Processing SPI
User Status change Event Orientation SPI
User IDGenerator
LDAP
Certificate
Custom Authentication Module

FIGURE 2–4 Authentication Service Components Within a Federated Access Manager Deployment

More information on the architecture of the Authentication Service can be found in the Authentication Service Architecture document on the OpenSSO web site.

## Policy Service

Authorization is the process with which Federated Access Manager evaluates the policies associated with an authenticated user's identity, and determines whether the user has permission to access a protected resource. (A *policy* defines the rules that specify a user's access privileges to a protected resource.) The Policy Service provides the authorization functionality using a rules-based engine. It interacts with the Federated Access Manager configuration data store, a delegation plug-in (which helps to determine the administrator's scope of privileges), and Identity Repository Service plug-ins to verify that the user has access privileges from a recognized authority. Policy can be configured using the administration console, and comprises the following:

- A *Schema* for the policy type (normal or referral) that describes the syntax of policy.
- A *Rule* which defines the policy itself and is made up of a *Resource*, an *Action* and a *Value*.
- *Condition(s)* to define constraints on the policy.
- *Subject(s)* to define the user or collection of users which the policy affects.
- A *ResponseProvider(s)* to send requested attribute values, typically based on the user profile, with the policy decision.

Figure 2–5 illustrates how the local and remote components of the Policy Service interact within a Federated Access Manager deployment. Note that the `PolicyServiceRequestHandler` maps to the `PolicyRequest` XML element.

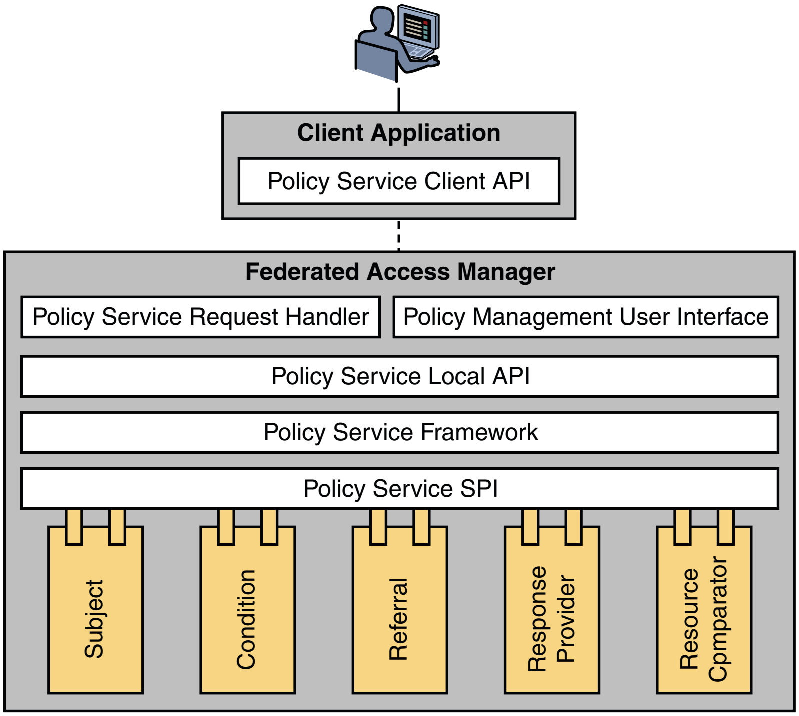

Client Application
Policy Service Client API
Federated Access Manager
Policy Service Request Handler
Policy Management User Interface
Policy Service Local API
Policy Service Framework
Policy Service SPI
Subject
Condition
Referral
Response Provider
Resource Cpmparator

**FIGURE 2–5** Policy Service Components within a Federated Access Manager Deployment

Policy agents are an integral part of authorization. They are programs, available for installation separate from Federated Access Manager, that police the web container which hosts the protected resources. When a user requests access to the protected resource (such as a server or an application), the policy agent intercepts the request and redirects it to the Federated Access Manager Authentication Service. Following authentication, the policy agent will enforce the authenticated user's assigned policies. Federated Access Manager supports two types of policy agents:

- The *web agent* is written in C and can protect any URL-based resource.
- The *Java Platform, Enterprise Edition (Java EE) agent* enforces URL-based policy and Java EE-based policy for Java applications on Java EE containers.

---

**Note –** When policy agents are implemented, all HTTP requests are implicitly denied unless explicitly allowed by the presence of two things:

1. A valid session
2. A policy allowing access

**Note –** If the resource is in the Not Enforced list defined for the policy agent, access is allowed even if there is no valid session.

More information on the architecture of the Policy Service can be found in the Policy Service Architecture document on the OpenSSO web site. For an overview of the available policy agents and links to specific information on installation, see the *Sun Java System Federated Access Manager Policy Agent 3.0 User's Guide*.

## Session Service

The mission of the Federated Access Manager Session Service is to maintain information about an authenticated user's session across all web applications participating in a user session. Additionally, Federated Access Manager provides continuous proof of the user's identity, enabling the user to access multiple enterprise resources without having to provide credentials each time. This enables the following types of user sessions.

- **Basic user session.** The user provides credentials to log in to one application, and then logs out of the same application.
- **SSO session.** The user provides credentials once, and then accesses multiple applications within the same DNS domain.
- **Cross domain SSO (CDSSO) session.** The user provides credentials once, and then accesses applications among multiple DNS domains.

A *user session* is the interval between the time a user attempts authentication through Federated Access Manager and is issued a session token, and the moment the session expires, is terminated by an administrator, or the user logs out. In what might be considered a typical user session, an employee accesses the corporate benefits administration service. The service, monitored by Federated Access Manager, prompts the user for a username and password. With the credentials Federated Access Manager can *authenticate*, or verify that the user is who he says he is. Following authentication, Federated Access Manager allows the user access to the service providing authorization is affirmed. Successful authentication through Federated Access Manager results in the creation of a *session data structure* for the user or entity by the Session Service. Generally speaking, the Session Service performs some or all of the following:

- Generates unique session identifiers, one for each user's session data structure

**Note –** A session data structure is initially created in the `INVALID` state with default values for certain attributes and an empty property list. Once the session is authenticated, the session state is changed to `VALID` and session data is updated with the user's identity attributes and properties.

- Maintains a master copy of session state information

---

**Note –** The session state maintained on the client side is a cached view of the actual session data structure. This cache can be updated by either the active polling mechanism or the session notification triggered by the Session Service.

---

- Implements time-dependent behavior of sessions — for example, enforces timeout limits
- Implements session life cycle events such as logout and session destruction
- Notifies all participants in the same SSO environment of session state changes
- Enables SSO and cross-domain single sign-on (CDSSO) among applications external to Federated Access Manager by providing continued proof of identity.
- Allows participating clients to share information across deployments
- Implements high availability facilities

Figure 2–6 illustrates the interactions between the local and remote components of the Session Service within a Federated Access Manager deployment.

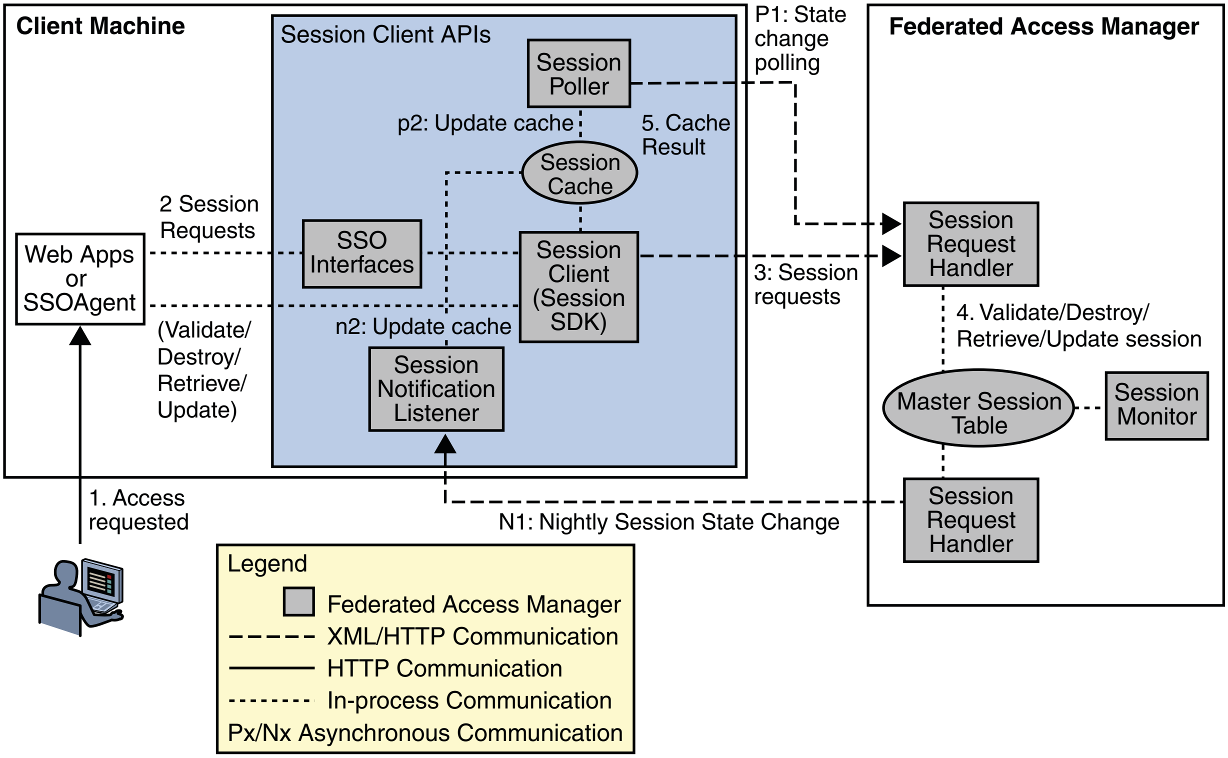

Client Machine
Session Client APIs
Session Poller
P1: State change polling
Federated Access Manager
p2: Update cache
5. Cache Result
Session Cache
2 Session Requests
SSO Interfaces
Session Client (Session SDK)
Session Request Handler
Web Apps or SSOAgent
3: Session requests
n2: Update cache
4. Validate/Destroy/ Retrieve/Update session
(Validate/ Destroy/ Retrieve/ Update)
Session Notification Listener
Master Session Table
Session Monitor
1. Access requested
N1: Nightly Session State Change
Session Request Handler
Legend
Federated Access Manager
XML/HTTP Communication
HTTP Communication
In-process Communication
Px/Nx Asynchronous Communication

FIGURE 2–6 Session Service Components within a Federated Access Manager Deployment

Additionally, Figure 2–7 illustrates how the messaging capabilities of Message Queue can be used to push session information to a persistent store based on the Berkeley DataBase (DB). Using Federated Access Manager in this manner enables the following key feature:

- Session Failover allows an alternative Federated Access Manager server to pick up a given user session when the server owning the original session fails.
- Session Constraints allow deployments to specify constraints on a sessions, such as one session per user.

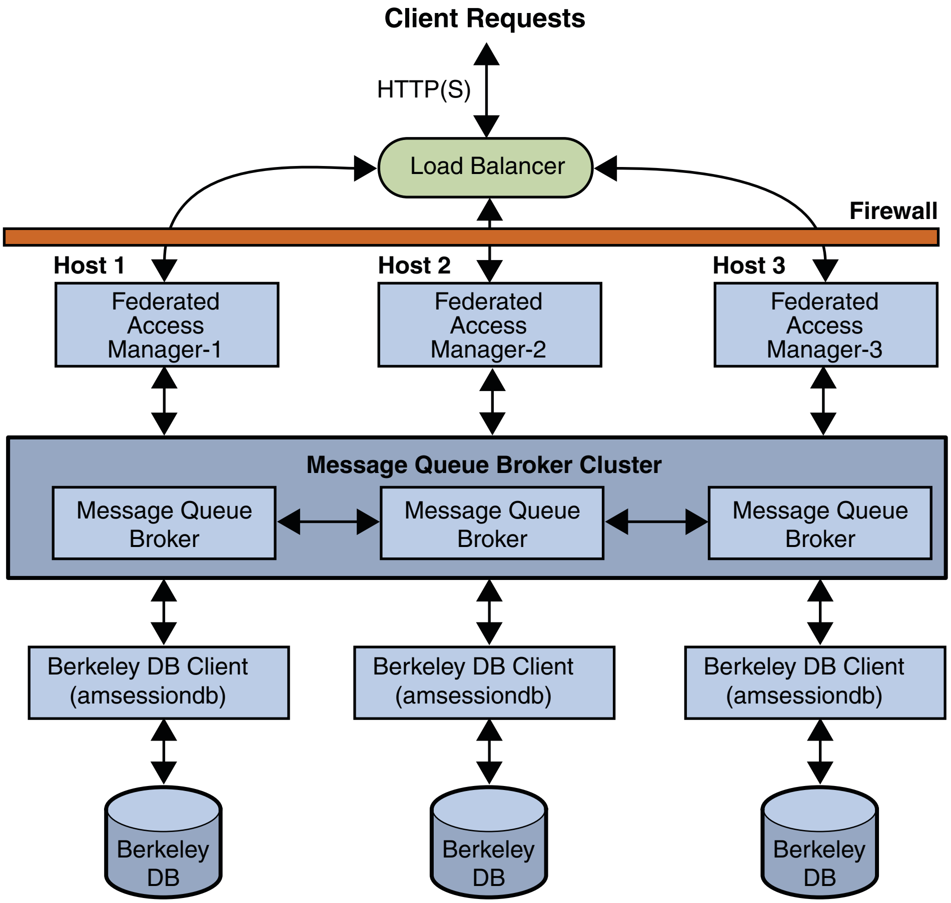

Client Requests
HTTP(S)
Load Balancer
Firewall
Host 1
Host 2
Host 3
Federated Access Manager-1
Federated Access Manager-2
Federated Access Manager-3
Message Queue Broker Cluster
Message Queue Broker
Message Queue Broker
Message Queue Broker
Berkeley DB Client (amsessiondb)
Berkeley DB Client (amsessiondb)
Berkeley DB Client (amsessiondb)
Berkeley DB
Berkeley DB
Berkeley DB

**FIGURE 2–7** Session Persistence Deployment Architecture

More information on the architecture of the Session Service can be found in the Session Service Architecture document on the OpenSSO web site. For more information on session failover, see Chapter 5, "Implementing Federated Access Manager Session Failover," in *Sun Federated Access Manager 8.0 Installation and Configuration Guide*.

## Logging Service

When a user logs in to a resource protected by Federated Access Manager, the Logging Service records information about the user's activity. The common Logging Service can be invoked by components residing on the same server as Federated Access Manager as well as those on the client machine, allowing the actual mechanism of logging (such as destination and formatting) to be separated from the contents which are specific to each component. You can write custom log operations and customize log plug-ins to generate log reports for specific auditing purposes.

Administrators can control log levels, authorize the entities that are allowed to create log entries and configure secure logging (the latter for flat files only). Logged information includes the name of the host, an IP address, the identity of the creator of the log entry, the activity itself, and the like. Currently, the fields logged as a *log record* are controlled by the Configurable Log Fields selected in the Logging Configuration page located under the System tab of the Federated Access Manager console. The Logging Service is dependent on the client application (using the Logging APIs) creating a programmatic `LogRecord` to provide the values for the log record fields. The logging interface sends the logging record to the Logging Service which determines the location for the log record from the configuration. A list of active logs can also be retrieved using the Logging API. Figure 2–8 illustrates the interactions between the local and remote components of the Logging Service in a Federated Access Manager deployment.

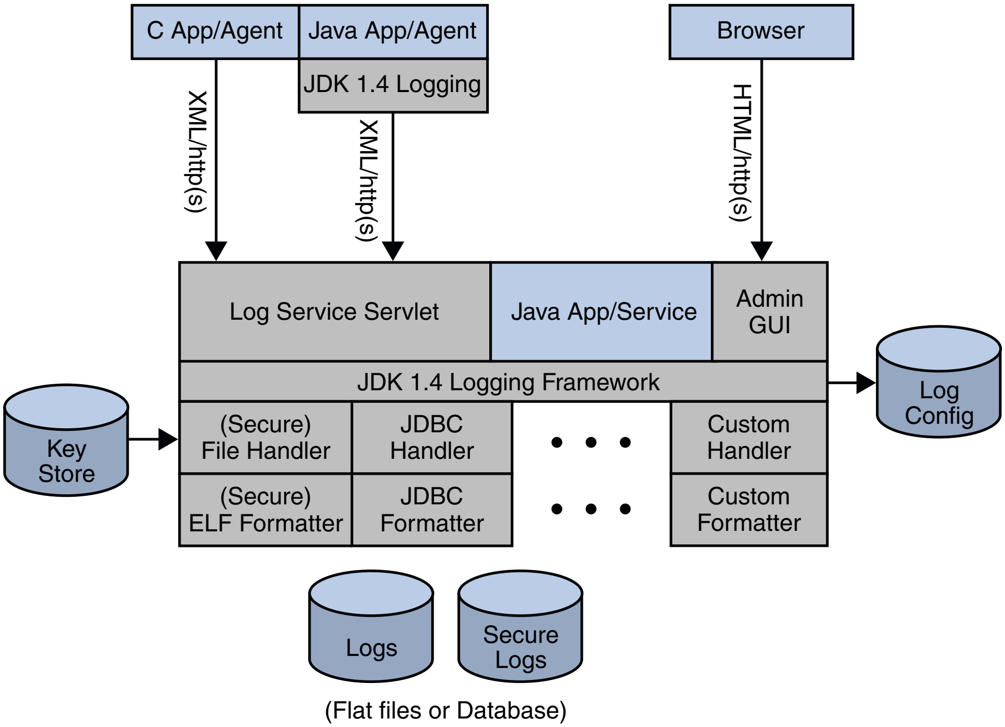

C App/Agent
Java App/Agent
JDK 1.4 Logging
Browser
XML/http(s)
XML/http(s)
HTML/http(s)
Log Service Servlet
Java App/Service
Admin GUI
JDK 1.4 Logging Framework
(Secure) File Handler
JDBC Handler
Custom Handler
(Secure) ELF Formatter
JDBC Formatter
Custom Formatter
Key Store
Log Config
Logs
Secure Logs
(Flat files or Database)

**FIGURE 2–8** Logging Service Components within a Federated Access Manager Deployment

**Caution –** Generally speaking, writing log records can be done remotely, using the Client SDK, but anything involving the reading API can only be done on the machine on which Federated Access Manager is installed. Using the reading API uses a great deal of system resources, especially when database logging is involved.

Logs can be written to flat files or to one of the supported databases (Oracle and MySQL). See Chapter 14, "Logging and the Java Enterprise System Monitoring Framework," for more information.

## Identity Repository Service

The Identity Repository Service allows Federated Access Manager to integrate an existing user data store (such as a corporate LDAP server) into the environment. The Identity Repository Service is able to access user profiles (as well as group and role assignments if supported) and is capable of spanning multiple repositories — even of different types. The Identity Repository Service is configured per realm under the Data Stores tab and its main functions are:

- To specify an identity repository that will store service configurations and attributes for users, groups and roles.
- To provide a list of identity repositories that can provide user attributes to the Policy Service and Federation Services frameworks.
- To combine the attributes obtained from different repositories.
- To provide interfaces to create, read, edit, and delete identity objects such as a realm, role, group, user, and agent.
- To map identity attributes using the Principal Name from the `SSOToken` object.

Access to the Identity Repository Service is provided by the `com.sun.identity.idm` Java package. The `AMIdentityRepository` class represents a realm that has one or more identity repositories configured and provides interfaces for searching, creating and deleting identities. The `AMIdentity` class represents an individual identity such as a user, group or role and provides interfaces to set, modify and delete identity attributes and assign and unassign services. `IdRepo` is an abstract class that contains the methods that need to be implemented by plug-ins when building new adapters for repositories not currently supported. The current implementation supports Sun Java System Directory Server, IBM Tivoli Directory and Microsoft Active Directory. Figure 2–9 illustrates the design of the Identity Repository Service.

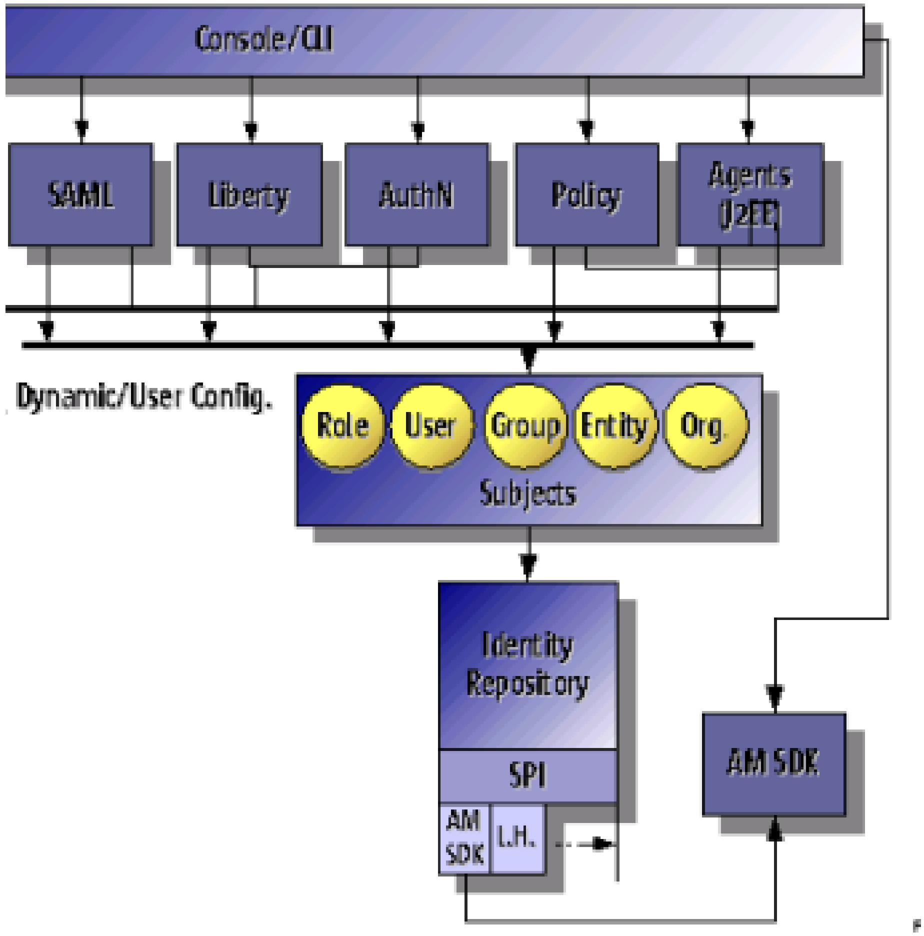

Console/CLI
SAML
Liberty
AuthN
Policy
Agents (J2EE)
Dynamic/User Config.
Role
User
Group
Entity
Org.
Subjects
Identity Repository
SPI
AM SDK
L.H.
AM SDK

FIGURE 2–9 Identity Repository Service Design

---

**Note –** Administrator roles are also defined by the Identity Repository Service. (This is currently available only when the Sun Directory Server With FAM Core Services schema is loaded.) For example, the Realm Administrator can access all data in all configured realms while the Subrealm Administrator can access data only within the specified realm. For more information, see XXXXXXX Deployment Planning Guide. For information on realm privileges, see "Privileges" in *Sun Federated Access Manager 8.0 Administration Guide*.

---

# Federation Services

Federated Access Manager provides an open and extensible framework for identity federation and associated web services that resolve the problems of identity-enabling web services, web service discovery and invocation, security, and privacy. Federation Services are built on the following standards:

- Liberty Alliance Project Identity Federation Framework (Liberty ID-FF) 1.1 and 1.2
- OASIS Security Assertion Markup Language (SAML) 1.0 and 1.1
- OASIS Security Assertion Markup Language (SAML) 2.0
- WS-Federation (Passive Requestor Profile)

Federation Services allows organizations to share identity information (for example, which organizations and users are trusted, and what types of credentials are accepted) securely. Once this is enabled securely, federating identities is possible — allowing a user to consolidate the many local identities configured among multiple service providers. With one federated identity, the user can log in at one identity provider's site and move to an affiliated site without having to re-establish identity. Figure 2–10 illustrates the interactions between local and remote components of the Federation Services in a Federated Access Manager deployment.

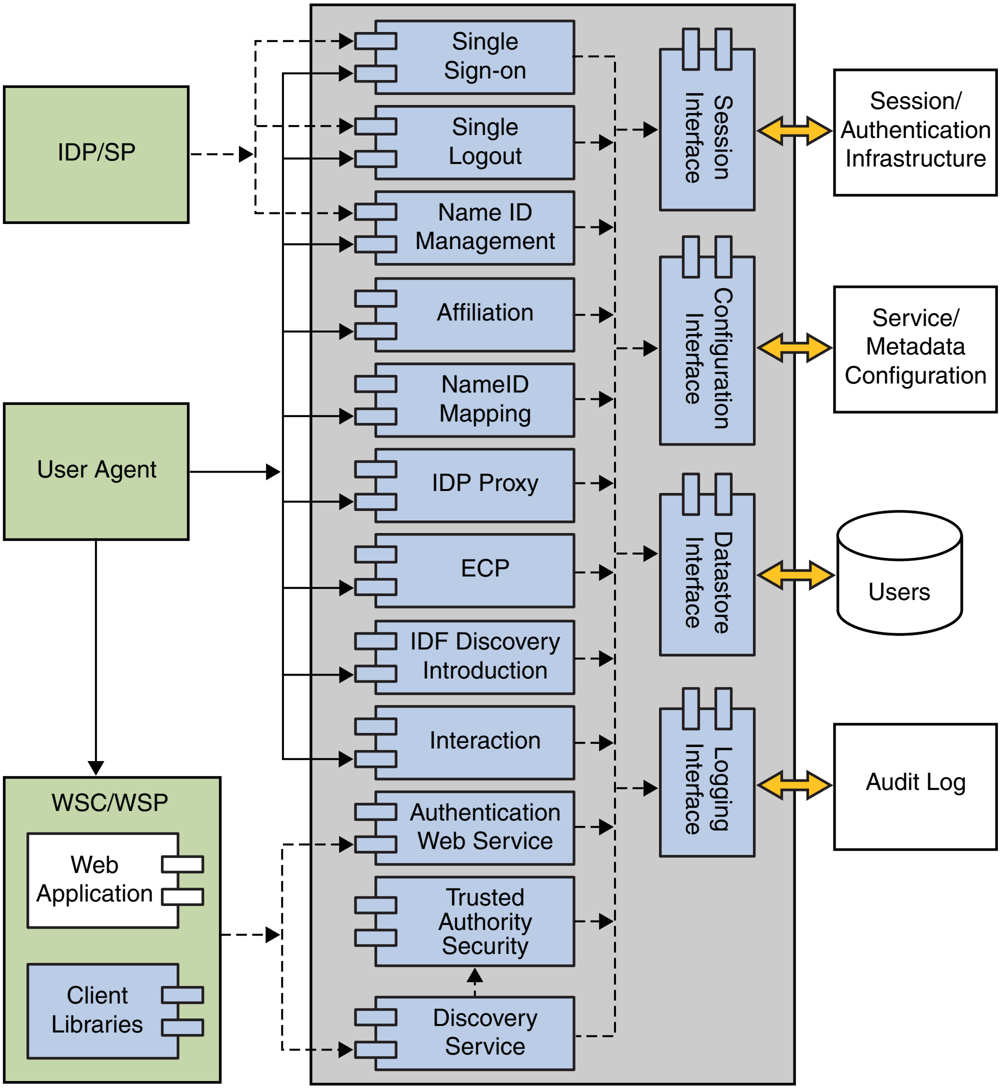

IDP/SP
User Agent
WSC/WSP
Web Application
Client Libraries
Single Sign-on
Single Logout
Name ID Management
Affiliation
NameID Mapping
IDP Proxy
ECP
IDF Discovery Introduction
Interaction
Authentication Web Service
Trusted Authority Security
Discovery Service
Session Interface
Configuration Interface
Datastore Interface
Logging Interface
Session/ Authentication Infrastructure
Service/ Metadata Configuration
Users
Audit Log

**FIGURE 2–10** Federation Services Components within a Federated Access Manager Deployment

More information on the Federation Services can be found in the Open Federation Architecture and the Federation Use Case documentation on the OpenSSO web site. Also, see Part III in this book.

## Web Services Stack

The Federated Access Manager Web Services Stack follows a standardized way of integrating web-based applications using XML, SOAP, and other open standards over an Internet Protocol (IP) backbone. They enable applications from various sources to communicate with each other because they are not tied to any one operating system or programming language. Businesses use web services to communicate with each other and their respective clients without having to know detailed aspects of each other's IT systems. Federated Access Manager provides web services that primarily use XML and SOAP over HTTP. These web services are designed to be centrally provided in an enterprise's network for convenient access by client applications. Federated Access Manager implements the follow web service specifications.

- Liberty Alliance Project Identity Web Services Framework (Liberty ID-WSF) 1.0, 1.1, and 2.0
- Web Services-Interoperability (WS-I) Basic Security Profile

The following table lists the Federated Access Manager web services.

**TABLE 2–1** Federated Access Manager Web Services Stack

| Web Service Name | Description |
|---|---|
| Authentication Web Service | Provides authentication to a web service client (WSC), allowing the WSC to obtain security tokens for further interactions with other services at the same provider. Upon successful authentication, the final Simple Authentication and Security Layer (SASL) response contains the resource offering for the Discovery Service. |
| Discovery Service | A web service that allows a requesting entity, such as a service provider, to dynamically determine a principal's registered attribute provider. Typically, a service provider queries the Discovery Service, which responds by providing a resource offering that describes the requested attribute provider. The implementation of the Discovery Service includes Java and web-based interfaces. |
| Liberty Personal Profile Service | A data service that supports storing and modifying a principal's identity attributes. Identity attributes might include information such as first name, last name, home address, and emergency contact information. The Liberty Personal Profile Service is queried or updated by a WSC acting on behalf of the principal. |

**TABLE 2–1** Federated Access Manager Web Services Stack *(Continued)*

| Web Service Name | Description |
|---|---|
| Security Token Service | The centralized Security Token Service that issues, renews, cancels, and validates security tokens. |
| SOAP Binding Service | A set of Java APIs implemented by the developer of a Liberty-enabled identity service. The APIs are used to send and receive identity-based messages using SOAP, an XML-based messaging protocol. |

Federated Access Manager uses both XML files and Java interfaces to manage web services and web services configuration data. A Federated Access Manager XML file is based on the structure defined in the Federated Access Manager Document Type Definition (DTD) files located in *path-to-context-root*`/fam/WEB-INF`. The main `sms.dtd` file defines the structure for all Federated Access Manager service files (located in *path-to-context-root*`/fam/WEB-INF/classes`).

**Caution –** Do not modify any of the DTD files. The Federated Access Manager API and their internal parsing functions are based on the default definitions and alterations to them may hinder the operation of the application.

For more information, see Part IV.

## Web Services Security

In message security, security information is applied at the message layer and travels along with the web services message. Message layer security differs from transport layer security in that it can be used to decouple message protection from message transport so that messages remain protected after transmission, regardless of how many hops they travel on. This message security is available as Web Services Security in Federated Access Manager and through the installation of an *authentication agent*. Web Services Security is the implementation of the WS-Security specifications and the Liberty Alliance Project Identity Web Services Framework (Liberty ID-WSF). Web Services Security allows communication with the Security Token Service to insert security tokens in outgoing messages and evaluate incoming messages for the same. Towards this end, authentication agents based on the Java Specification Request (JSR) 196 must be downloaded and installed on the web services client (WSC) machine and the web services provider (WSP) machine.

**Note –** JSR 196 agents can be used only on Sun Java System Application Server or Glassfish web containers.

To secure web services communications, the requesting party must first be authenticated with a security token which is added to the SOAP header of the request. Additionally, the WSC needs to be configured to supply message level security in their SOAP requests and the WSP needs to be configured to enable message level security in their SOAP responses. Figure 2–11 illustrates the components used during a secure web services interaction.

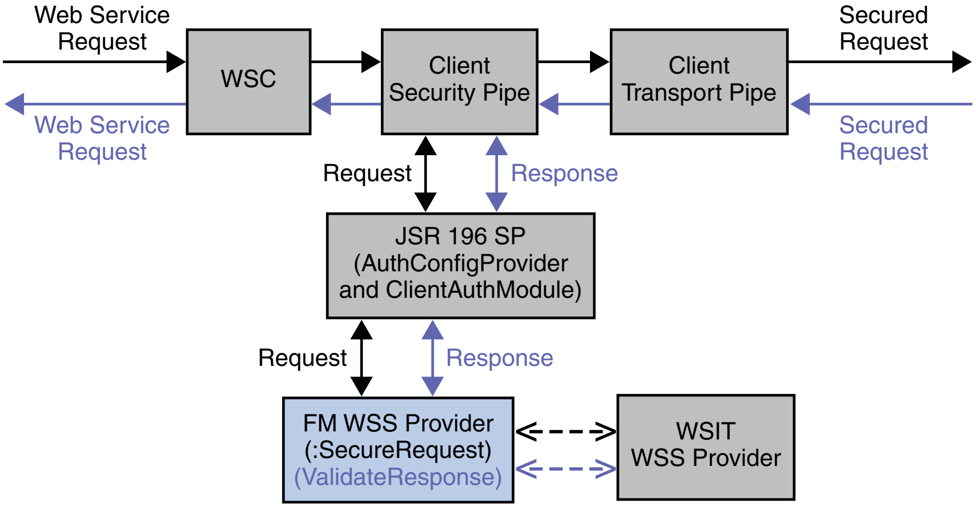

Web Service Request
WSC
Client Security Pipe
Client Transport Pipe
Secured Request
Web Service Request
Secured Request
Request
Response
JSR 196 SP (AuthConfigProvider and ClientAuthModule)
Request
Response
FM WSS Provider (:SecureRequest) (ValidateResponse)
WSIT WSS Provider

FIGURE 2–11 Web Services Security Components within a Federated Access Manager Deployment

---

**Note –** The stand alone applications can directly invoke the interfaces (secure request by WSC, and validate response by WSP) from the WS-Security Library and establish message-level end-to-end web service security. Standalone Java applications do not need the WS-Security Provider Plugin.

---

For more information, see Part IV.

# Identity Web Services

Federated Access Manager contains client interfaces for authentication, authorization, session management, and logging in Java, C, and C++ (using a proprietary XML and SOAP over HTTP or HTTPs communication). These interfaces are used by policy agents and custom applications. Development using these interfaces, though, is labor-intensive. Additionally, the interfaces cause dependencies on Federated Access Manager. Therefore, Federated Access Manager has now implemented simple interfaces that can be used for:

- Authentication (verification of user credentials, password management)
- Authorization (policy evaluation for access to secured resources)
- Provisioning (self-registration, creating or deleting identity profiles, retrieve or update identity profile attributes)
- Token validation
- Search (return a list of identity profile attributes that match a search filter)

---

**Note –** Identity Web Services also interact with the Logging Service to audit and record Identity Web Services interactions.

---

These Identity Services are offered using either SOAP and the Web Services Description Language (WSDL) or Representational State Transfer (REST). With SOAP Identity Web Services, you point an integrated development environment (IDE) application project to the appropriate URL and generate the stub code that wraps the function calls to the services. (You can also use `wscompile`.) With REST Identity Web Services, no coding is necessary. It works right out of box.

---

**Note –** Federated Access Manager supports Eclipse, NetBeans, and Visual Studio®.

---

When Identity Web Services have been implemented, a user interacts with the application which calls the identity repository to retrieve user profile data for authentication and personalization, the configuration data store to retrieve policy data for authorization, and the audit repository for log requests. The application authenticates, authorizes, audits, and finally creates personalized services for the user by calling either the SOAP/WSDL or REST Identity Web Service as provided by Federated Access Manager.

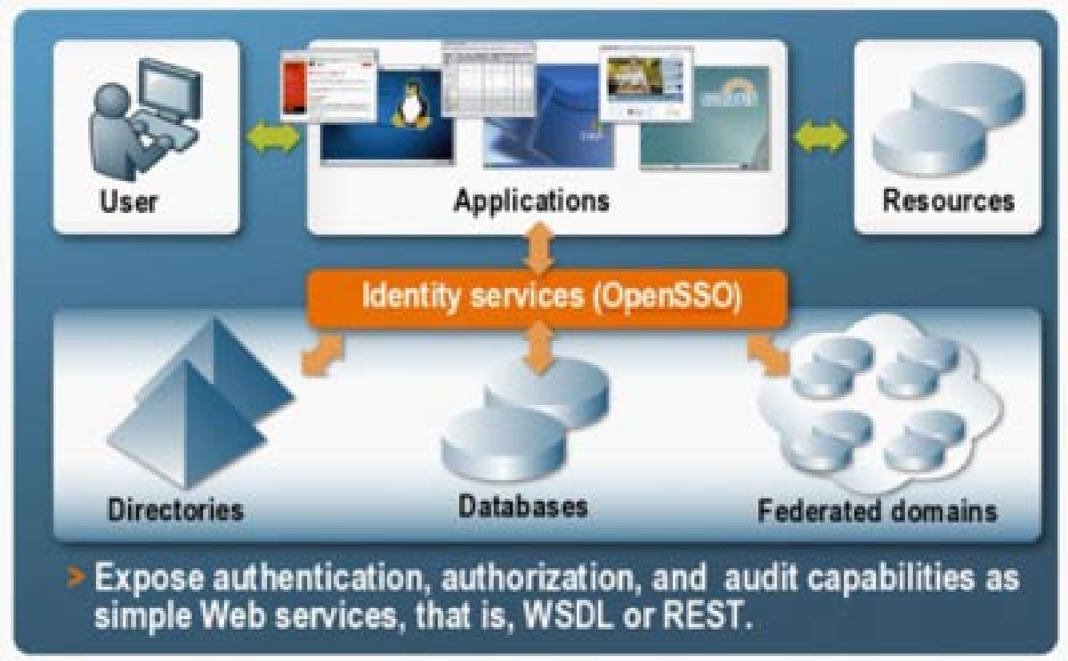

User
Applications
Resources
Identity services (OpenSSO)
Directories
Databases
Federated domains
Expose authentication, authorization, and audit capabilities as simple Web services, that is, WSDL or REST.

**FIGURE 2–12** Basic Identity Web Services Process

For more information, see Part IV.

# Global Services

Global services take configuration values and perform functions for Federated Access Manager on a global basis. The following table lists the global services with brief descriptions.

**TABLE 2–2** Global Federated Access Manager Services

| Service | What it Does |
|---|---|
| Common Federation Configuration | Contains configuration attributes for Federation Services. |
| Liberty ID-FF Service Configuration | Contains configuration attributes for the Liberty Alliance Project Identity Federation Framework. |
| Liberty ID-WSF Security Service | Contains configuration attributes for the Liberty Alliance Project Identity Web Services Framework. |

**TABLE 2–2** Global Federated Access Manager Services *(Continued)*

| Service | What it Does |
|---|---|
| Liberty Interaction Service | Contains configuration attributes for the Liberty Alliance Project Interaction Service — used to get consent from an owner to expose data, or to get additional data. |
| Multi-Federation Protocol | Contains configuration attributes for multi-federation protocol circles of trust. |
| Password Reset | Contains configuration attributes for the Password Reset Service. |
| Policy Configuration | Contains configuration attributes for the Policy Service. |
| SAML v2 Service Configuration | Contains configuration attributes for the SAML v2 interactions. |
| SAML v2 SOAP Binding | Contains configuration attributes for SAML v2 SOAP Binding Service. |
| Security Token Service | Contains configuration attributes for the Security Token Service. |
| Session | Contains configuration attributes for the Session Service. |
| User | Contains configuration attributes for user profiles. |

# Realms

A *realm* is the unit that Federated Access Manager uses to organize configuration information. Authentication properties, authorization policies, data stores, subjects (including a user, a group of users, or a collection of protected resources) and other data can be defined within the realm. The data stored in a realm can include, but is not limited to:

- One or more subjects (a user, a group of users, or a collection of protected resources)
- A definition of one or more data stores to store subject (user) data
- Authentication details identifying, for example, the location of the authentication repository, and the type of authentication required.
- Policy information that will be used to determine which resources protected by Federated Access Manager the subjects can access.
- Responder configurations that allows applications to personalize the user experience, once the user has successfully authenticated and been given access.
- Administration data for realm management

You create a top-level realm when you deploy Federated Access Manager. The top-level realm (by default `opensso`) is the root of the Federated Access Manager instance and contains Federated Access Manager configuration data; it cannot be changed after it is created. In general, you should use the default root realm to configure identity data stores, and manage policies and authentication chains. During deployment, Federated Access Manager creates a Realm Administrator who can perform all operations in the configured root realm, and a Policy Administrator who can only create and manage policies.

All other realms are configured under the `opensso` realm. These sub-realms may contain other sub-realms and so on. Sub-realms identify sets of users and groups that have different authentication or authorization requirements. The use of sub-realms should be restricted to the following two scenarios.

1. **Application Policy Delegation** The use case for this is when you need to have different Policy Administrators to create policies for a sub-set of resources. For example, let's assume a sub-realm is created and named Paycheck. This sub-realm is configured with a policy referral from the root realm for configuring protection of resources starting with https://paycheck.sun.com/paycheck. Within the Paycheck sub-realm, a Paycheck Administrator role or group is created and assigned Policy Administration privileges. These administrators are now able to login to the sub-realm and create policies for their applications. By default, the sub-realm inherits the same configuration data store and authentication chains configured for its parent; if these configurations change in the parent, a corresponding change would be needed in the sub-realm. Additionally, all users will still log in to the root realm for access to all the applications. The sub-realm is primarily for the Policy Administrator to manage policies for the application. An educated guess on the number of sub-realms that can be supported would be about 100.
2. **ISP/ASP/Silo** The use case for this scenario is when each sub-realm is to have its own set of identity data stores, authentication chains, and policies. Ideally the only common thread between the root and the sub-realm would be the referral policy created in the root realm to delegate a set of resources to the sub-realm. Users would not be able to log in to the root realm (unless they are a member) but would have to authenticate to their sub-realm. Also, agents would have to be configured to redirect user authentication to the particular sub-realm. With regards to performance, the most resource consuming component would be when persistent searches created by the data stores connect to the same directory. An educated guess on the number of sub-realms that can be supported would be about 50.

The Federated Access Manager framework aggregates realm properties as part of the configuration data. Figure 2–13 illustrates how configuration data can use a hierarchy of realms to distribute administration responsibilities. Region 1, Region 2, and Region 3 are realms; Development, Operations, and Sales are realms sub to Region 3.

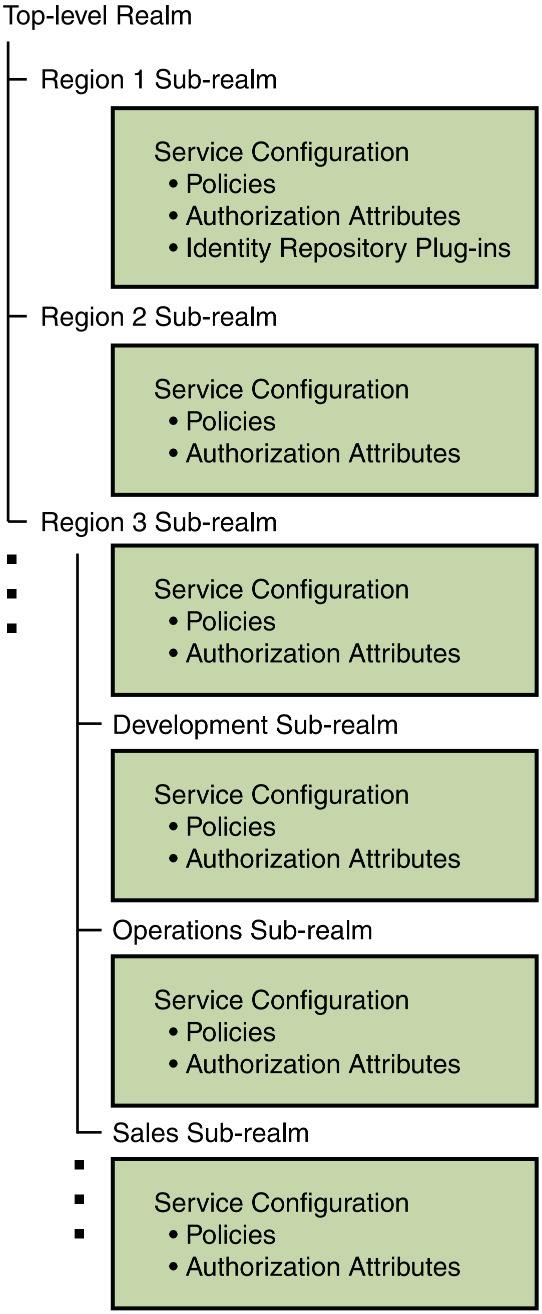

Top-level Realm
Region 1 Sub-realm
Service Configuration
• Policies
• Authorization Attributes
• Identity Repository Plug-ins
Region 2 Sub-realm
Service Configuration
• Policies
• Authorization Attributes
Region 3 Sub-realm
Service Configuration
• Policies
• Authorization Attributes
Development Sub-realm
Service Configuration
• Policies
• Authorization Attributes
Operations Sub-realm
Service Configuration
• Policies
• Authorization Attributes
Sales Sub-realm
Service Configuration
• Policies
• Authorization Attributes

**FIGURE 2–13** Realm Hierarchy for Configuration Data

---

**Note –** Federated Access Manager 8.0 supports the Sun Java System Access Manager Legacy mode (which contains no realms) with a provided interface.

---

# Additional Components

The following sections provide information on additional components used in a Federated Access Manager deployment.

- "Data and Data Stores" on page 54
- "The `bootstrap` File" on page 60
- "Policy Agents" on page 61
- "Authentication Agents" on page 62
- "Federated Access Manager Tools" on page 62
- "Client SDK" on page 63
- "Service Provider Interfaces for Plug-ins" on page 63

## Data and Data Stores

Federated Access Manager services need to interact with a number of different data stores. The following distinct repositories can be configured.

- A configuration repository provides server and service specific data.
- One or more identity repositories provide user profile information.
- Authentication repositories provide authentication credentials to a particular module of the Authentication Service.

A common LDAP connection pooling facility allows efficient use of network resources. In the simplest *demonstration* environment, a single LDAP repository is sufficient for all data however, the typical *production* environment tends to separate configuration data from other data. The following sections contain more specific information.

- "Configuration Data" on page 54
- "Identity Data" on page 57
- "Authentication Data" on page 60

### Configuration Data

The default configuration of Federated Access Manager creates a branch in a fresh installation of a configuration data store for storing service configuration data and other information pertinent to the server's operation. Federated Access Manager components and plug-ins access the configuration data and use it for various purposes including:

- Accessing policy data for policy evaluation.

- Finding location information for identity data stores and Federated Access Manager services.
- Retrieving authentication configuration information that define how users and groups authenticate.
- Finding which partner servers can send trusted SAML assertions.

Federated Access Manager supports Sun Java System Directory Server and the open source OpenDS as configuration data stores. Flat files (supported in previous versions of the product) are no longer supported but configuration data store failover is — using replication. Figure 2–14 illustrates how configuration data in the embedded data store is accessed.

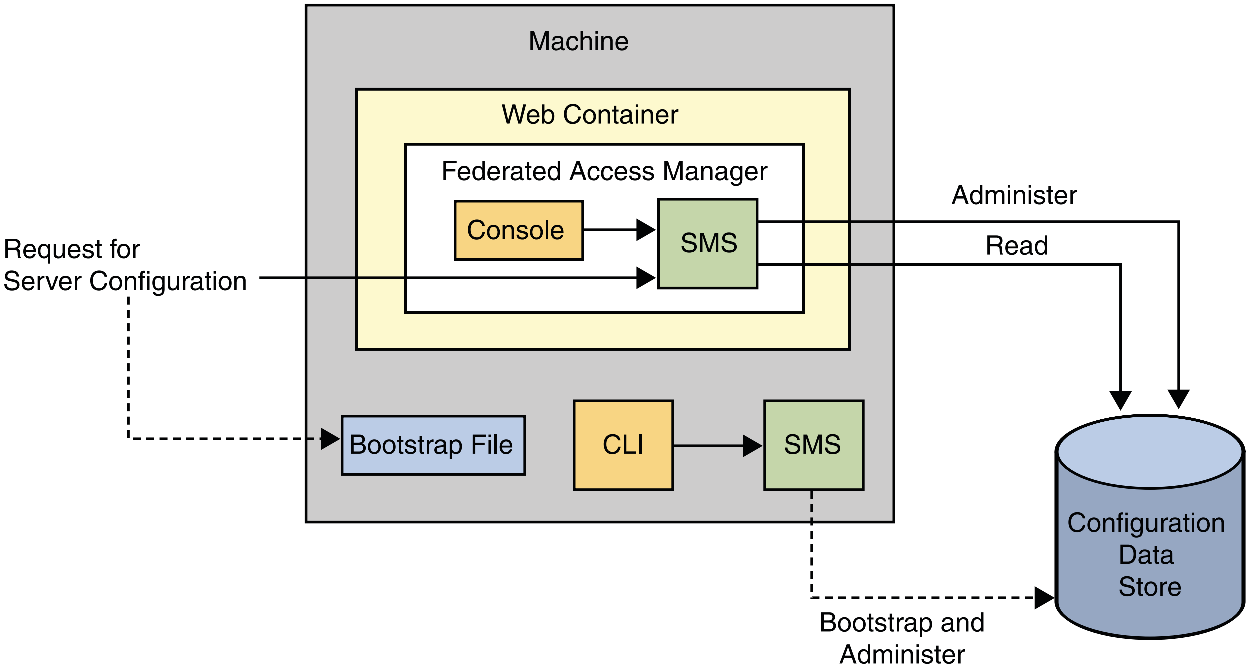

Machine
Web Container
Federated Access Manager
Console
SMS
Administer
Read
Request for Server Configuration
Bootstrap File
CLI
SMS
Configuration Data Store
Bootstrap and Administer

**FIGURE 2–14** Accessing Configuration Data

Previous releases of Access Manager and Federation Manager stored product configuration data in a property file named `AMConfig.properties` that was installed local to the product instance directory. This file is deprecated for Federated Access Manager on the server side although still supported for agents on the client side. See the *Sun Federated Access Manager 8.0 Installation and Configuration Guide* for more information.

Configuration data comprises the attributes and values in the Federated Access Manager configuration services, as well as default Federated Access Manager users like `amadmin` and `anonymous`. Following is a partial listing of the XML service files that contribute to the data. They can be found in the *path-to-context-root*`/fam/WEB-INF/classes` directory.

---

**Note –** The data in this node branch is private and is mentioned here for information purposes only.

---

- `AgentService.xml`
- `amAdminConsole.xml`
- `amAgent70.xml`
- `amAuth.xml`
- `amAuth-NT.xml`
- `amAuthAD.xml`
- `amAuthAnonymous.xml`
- `amAuthCert.xml`
- `amAuthConfig.xml`
- `amAuthDataStore.xml`
- `amAuthHTTPBasic.xml`
- `amAuthJDBC.xml`
- `amAuthLDAP.xml`
- `amAuthMSISDN.xml`
- `amAuthMembership.xml`
- `amAuthNT.xml`
- `amAuthRADIUS.xml`
- `amAuthSafeWord-NT.xml`
- `amAuthSafeWord.xml`
- `amAuthSecurID.xml`
- `amAuthWindowsDesktopSSO.xml`
- `amClientData.xml`
- `amClientDetection.xml`
- `amConsoleConfig.xml`
- `amDelegation.xml`
- `amEntrySpecific.xml`
- `amFilteredRole.xml`
- `amG11NSettings.xml`
- `amLogging.xml`
- `amNaming.xml`
- `amPasswordReset.xml`
- `amPlatform.xml`
- `amPolicy.xml`
- `amPolicyConfig.xml`
- `amRealmService.xml`
- `amSession.xml`
- `amUser.xml`
- `amWebAgent.xml`

- `idRepoEmbeddedOpenDS.xml`
- `idRepoService.xml`
- `identityLocaleService.xml`
- `ums.xml`

**Caution –** By default, the Federated Access Manager configuration data is created and maintained in the embedded configuration data store apart from any identity data. Although users can be created in the configuration data store this is only recommended for demonstrations and development environments.

For more information, see "Embedded Configuration Data" on page 67.

## Identity Data

An *identity repository* is a data store where information about users and groups in an organization is stored. User profiles can contain data such as a first name, a last name, a phone number, group membership, and an e-mail address; an identity profile template is provided out-of-the-box but it can be modified to suit specific deployments.

Identity data stores are defined per realm. Because more than one identity data store can be configured per realm Federated Access Manager can access the many profiles of one identity across multiple data repositories. Sun Java System Directory Server with Federated Access Manager Schema, Microsoft Active Directory, IBM Tivoli Directory and the AMSDK data store are the currently supported identity repositories. Plug-ins can be developed to integrate other types of repositories (for example, a relational database). Figure 2–15 illustrates a Federated Access Manager deployment where the identity data and the embedded configuration data are kept in separate data stores.

verify support officially on Tivoli (Replaces generic LDAP v3), AD, Sun DS with FAM schema and amsdk

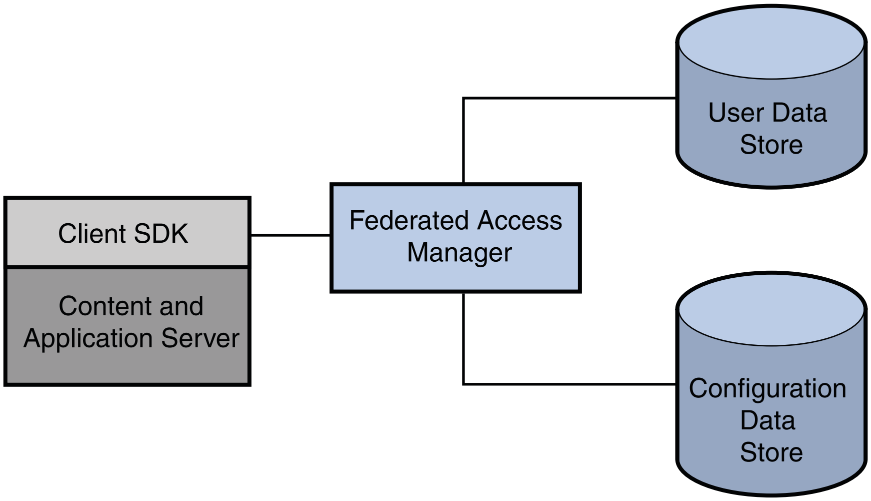


**FIGURE 2–15** Federated Access Manager Deployment with Two Data Stores

---

**Note –** The information in an identity repository is maintained by provisioning products separate from Federated Access Manager. The supported provisioning product is Sun Java System Identity Manager.

---

Federated Access Manager provides out-of-the-box plug-in support for some identity repositories. Each default plug-in configuration includes details about what operations are supported on the underlying data store. Once a realm is configured to use a plug-in, the framework can instantiate it and execute the operations on the appropriate identity repository. Each new plug-in developed must have a corresponding service management schema defining its configuration attributes. This schema would be integrated as a sub schema into `idRepoService.xml`, the service management file for the Identity Repository Service that controls the identity data stores available under a realm's Data Stores tab. The following sections contain information on the out-of-the-box plug-ins.

**Remark 2–1 Reviewer** **Please confirm these. I have gotten different comments from different reviewers.**

- “Generic Lightweight Directory Access Protocol (LDAP) version 3” on page 59
- “LDAPv3 Plug-in for Active Directory” on page 59
- “LDAPv3 Plug-in for Tivoli Directory” on page 59
- “Sun Directory Server With FAM Core Services” on page 59
- “Sun Directory Server With Full Schema (including Legacy)” on page 59
- “Access Manager Repository Plug-in” on page 59

### Generic Lightweight Directory Access Protocol (LDAP) version 3

The Generic LDAPv3 identity repository plug-in can reside on an instance of any directory that complies with the LDAPv3 specifications. The underlying directory cannot make use of features that are not part of the LDAP version 3 specification, and no specific DIT structure can be assumed as LDAPv3 identity repositories are simply DIT branches that contain user and group entries. Each data store has a name that is unique among a realm's data store names, but not necessarily unique across all realms. The `com.sun.identity.idm.plugins.ldapv3.LDAPv3Repo` class provides the default LDAPv3 identity repository implementation. There are also implementations for Active Directory and IBM Tivoli Directory

### LDAPv3 Plug-in for Active Directory

The Generic LDAPv3 identity repository plug-in was used to develop a default plug-in to write identity data to an instance of Microsoft® Active Directory®. The administration console provides a way to configure the directory but the schema needs to be loaded manually.

### LDAPv3 Plug-in for Tivoli Directory

The Generic LDAPv3 identity repository plug-in was used to develop a default plug-in to write identity data to an instance of IBM Tivoli Directory®. The administration console provides a way to configure the directory but the schema needs to be loaded manually. XXXXXX Where is this config doc'ed?XXXXX

### Sun Directory Server With FAM Core Services

This repository resides in an instance of Sun Java System Directory Server and holds the identity data. This option is available during the initial configuration of Federated Access Manager.

### Sun Directory Server With Full Schema (including Legacy)

This repository resides in an instance of Sun Java System Directory Server and holds the configuration data when installing Federated Access Manager in Legacy and Realm mode. This option must be manually configured.

### Access Manager Repository Plug-in

The Access Manager Repository can reside only in Sun Java System Directory Server and is used with the **Sun Directory Server With Access Manager Schema**. During installation, the repository is created in the same instance of Sun Java System Directory Server that holds the configuration data. The Access Manager Repository Plug-in is designed to work with Sun Java System Directory Server as it makes use of features specific to the server including *roles* and *class of service*. It uses a DIT structure similar to that of previous versions of Access Manager.

---

**Note –** This is no longer provided out of the box and many pieces are marked for deprecation. The Access Manager Repository is compatible with previous versions of Access Manager.

---

When you configure an instance of Access Manager in realm mode for the first time, the following occurs:

- An Access Manager Repository is created under the top-level realm.
- The Access Manager Repository is populated with internal Access Manager users.

---

**Note –** The Java Enterprise System installer does not set up an Access Manager Repository when you configure an Access Manager instance in legacy mode. Legacy mode requires an identity repository that is mixed with the Access Manager information tree under a single directory suffix.

---

### Authentication Data

Authentication data contains authentication credentials for Federated Access Manager users. An authentication data store is aligned with a particular authentication module, and might include:

- RADIUS servers
- SafeWord authentication servers
- RSA ACE/Server systems (supports SecurID authentication)
- LDAP directory servers

---

**Note –** Identity data may include authentication credentials although authentication data is generally stored in a separate authentication repository.

---

## The `bootstrap` File

Federated Access Manager uses a file to bootstrap itself. Previously, `AMConfig.properties` held configuration information to bootstrap the server but now a file named `bootstrap` points to the configuration data store allowing the setup servlet to retrieve the bootstrapping data. After deploying the Federated Access Manager WAR and running the configuration wizard, configuration data is written to the configuration data store by the service management API contained in the Java package, `com.sun.identity.sm`. The setup servlet creates `bootstrap` in the top-level configuration directory. The content in bootstrap can be either of the following:

- A directory local to Federated Access Manager (for example, `/export/SUNWam`) indicating the server was configured with a previous release. The directory is where `AMConfig.properties` resides.
- An encoded URL that points to a directory service using the following format:

`ldap://`*ds-host*`:`*ds-port*`/`*server-instance-name*`?pwd=`*encrypted-amadmin-password*`&`
`embeddedds=`*path-to-directory-service-installation*`&basedn=`*base-dn*`&`
`dsmgr=`*directory-admin*`&dspwd=`*encrypted-directory-admin-password*

For example:

```
ldap://ds.samples.com:389/http://owen2.red.sun.com:8080/fam?
pwd=AQIC5wM2LY4Sfcxi1dVZEdtfwar2vhWNkmS8&embeddedds=/fam/opends&
basedn=dc=fam,dc=java,dc=net&dsmgr=cn=Directory Manager
&dspwd=AQIC5wM2LY4Sfcxi1dVZEdtfwar2vhWNkmS8
```

where

- `ds.samples.com:389` is the host name and port of the machine on which the directory is installed.
- `http://owen2.red.sun.com:8080/fam` is the instance name.
- `AQIC5wM2LY4Sfcxi1dVZEdtfwar2vhWNkmS8` is the encrypted password of the OpenSSO administrator.
- `/fam/opends` is the path to the directory installation.
- `dc=fam,dc=java,dc=net` is the base DN.
- `cn=Directory Manager` is the directory administrator.
- `AQIC5xM2LY4SfcximdVZEdtfwar4vhWNkmG7` is the encrypted password for the directory administrator.

If more than one URL is present in the file and Federated Access Manager is unable to connect or authenticate to the data store at the first URL, the bootstrapping servlet will try the second (and so on). Additionally, the number sign [#] can be used to exclude a URL as in:

```
# ldap://ds.samples.com:389/http://owen2.red.sun.com:8080/fam?
pwd=AQIC5wM2LY4Sfcxi1dVZEdtfwar2vhWNkmS8&embeddedds=/fam/opends&
basedn=dc=fam,dc=java,dc=net&dsmgr=cn=Directory+Manager
&dspwd=AQIC5wM2LY4Sfcxi1dVZEdtfwar2vhWNkmS8
```

## Policy Agents

Policy agents are an integral part of SSO and CDSSO sessions. They are programs that police the web container on which resources are hosted. All policy agents interact with the Authentication Service in two ways:

- To authenticate itself in order to establish trust. This authentication happens using the Client SDK.
- To authenticate users having no valid session for access to a protected resource. This authentication happens as a browser redirect from the Distributed Authentication User Interface.

When a user requests access to a protected resource such as a server or an application, the policy agent intercepts the request and redirects it to the Federated Access Manager Authentication Service for authentication. Following this, the policy agent requests the authenticated user's assigned policy and evaluates it to allow or deny access. (A *policy* defines the rules that specify a user's access privileges to a protected resource.) Federated Access Manager supports two types of policy agents:

- The *web agent* enforces URL-based policy for C applications.
- The *Java EE agent* enforces URL-based policy and Java-based policy for Java applications on Java EE containers.

Both types of agents are available for you to install as programs separate from Federated Access Manager. Policy agents are basically clients written using the Client SDK and Identity Services.

---

**Note –** All HTTP requests are implicitly denied unless explicitly allowed by the presence of a valid session and a policy allowing access. If the resource is defined in the Not Enforced list for the policy agent, access is allowed even if there is no valid session.

---

For more information, see J2EE Agent Architecture and Web Agent and C-API Architecture on the OpenSSO web site. For an overview of the available policy agents and links to specific information on installation, see the *Sun Java System Federated Access Manager Policy Agent 3.0 User's Guide*.

## Authentication Agents

Authentication agents plug into web containers to provide message level security for web services, and supports both Liberty Alliance Project token profiles as well as Web Services-Interoperability Basic Security Profiles (WS-I BSP). (A *profile* defines the HTTP exchanges required to transfer XML requests and responses between web service clients and providers.) Authentication agents use an instance of Federated Access Manager for all authentication decisions. Web services requests and responses are passed to the appropriate authentication modules using standard Java representations based on the transmission protocol. An HTTP Authentication Agent or a SOAP Authentication Agent can be used. For more information, see "Web Services Security" on page 47.

## Federated Access Manager Tools

Contained within the Federated Access Manager ZIP are `famAdminTools.zip` and `famSessionTools.zip`. The following sections have some information about these tools.

- "`famadm` Command Line Interface" on page 63
- "Session Failover Tools" on page 63

### `famadm` Command Line Interface

`famadm`, the command line interface (CLI), provides a second option to administer Federated Access Manager using the command line. For example, `famadm` can be used to create a policy or import and export Liberty ID-FF metadata. `famadm` is the recommended way for batch processing. It is in `famAdminTools.zip`. For more information, see Chapter 4, “Installing the Federated Access Manager Utilities and Scripts,” in *Sun Federated Access Manager 8.0 Installation and Configuration Guide* and Part I, “Command Line Interface Reference,” in *Sun Federated Access Manager Administration Reference*.

### Session Failover Tools

`famSessionTools.zip` contains scripts and binaries for setting up session failover and databases. For more information, see Chapter 5, “Implementing Federated Access Manager Session Failover,” in *Sun Federated Access Manager 8.0 Installation and Configuration Guide*.

## Client SDK

Enterprise resources cannot be protected by Federated Access Manager until the Federated Access Manager Client SDK is installed on the machine that contains the resource that you want to protect. (The Client SDK is automatically installed with a policy agent.) The Client SDK allows you to customize an application by enabling communication with Federated Access Manager for retrieving user, session, and policy data. For more information, see Chapter 1, “Enhancing Remote Applications Using the Client Software Development Kit,” in *Sun Federated Access Manager 8.0 Developer’s Guide* and the *Federated Access Manager 8.0 Java API Reference*.

## Service Provider Interfaces for Plug-ins

The Federated Access Manager service provider interfaces (SPI) can be implemented as plug-ins to provide customer data to the Federated Access Manager framework for back-end processing. Some customer data comes from external data base applications such as identity repositories while other customer data comes from the Federated Access Manager plug-ins themselves. You can develop additional custom plug-ins to work with the SPI. For a complete list of the SPI, see the *Federated Access Manager 8.0 Java API Reference*. Additional information can be found in the *Sun Federated Access Manager 8.0 Developer’s Guide*. The following sections contain brief descriptions.

- “Authentication Service SPI” on page 64
- “Federation Service SPI” on page 64
- “Identity Repository Service SPI” on page 64
- “Policy Service SPI” on page 64
- “Service Configuration Plug-in” on page 64

## Authentication Service SPI

The `com.sun.identity.authentication.spi` package provides interfaces and classes for writing a supplemental authentication module to plug into Federated Access Manager. The `com.sun.identity.authentication` package provides interfaces and classes for writing a remote client application that can access user data in a specified identity repository to determine if a user's credentials are valid.

## Federation Service SPI

The `com.sun.identity.federation.services` package provides plug-ins for customizing the Liberty ID-FF profiles implemented by Federated Access Manager. The `com.sun.identity.federation.plugins` package provides an interface that can be implemented to perform user specific processing on the service provider side during the federation process. The `com.sun.identity.saml2.plugins` package provides the SAML v2 service provider interfaces (SPI). The `com.sun.identity.wsfederation.plugins` package provides the WS-Federation based SPI.

## Identity Repository Service SPI

The `com.sun.identity.idm` package contains the `IdRepo` interface that defines the abstract methods which need to be implemented or modified by Identity Repository Service plug-ins. The `com.sun.identity.plugin.datastore` package contains interfaces that search for and return identity information such as user attributes and membership status for purposes of authentication.

## Policy Service SPI

The `com.sun.identity.policy.interfaces` package provides interfaces for writing custom policy plug-ins for Conditions, Subjects, Referrals, Response Providers and Resources.

## Service Configuration Plug-in

The `com.sun.identity.plugin.configuration` package provides interfaces to store and manage configuration data required by the core Federated Access Manager components and other plug-ins.

---

**Note –** In previous releases, the functionality provided by the Service Configuration plug-in was known as the Service Management Service (SMS).

---

◆ ◆ ◆

# CHAPTER 3

# Simplifying Federated Access Manager

This chapter contains information on the usability and manageability features of Federated Access Manager. It includes the following sections:

- "Installation and Configuration" on page 65
- "Embedded Configuration Data" on page 67
- "Centralized Agent Configuration" on page 68
- "Common Tasks" on page 70
- "Third Party Integration" on page 71

## Installation and Configuration

Previous versions of Sun Microsystems' access management server product were built for multiple hardware platforms, and different web containers. The complexity of this development process led to the release of separate platform and container patches. To alleviate this extraneous development, Federated Access Manager is now available as a single ZIP file which can be downloaded, unzipped, and quickly deployed; there will be no separate installations for each hardware platform. The ZIP file will contain the full Federated Access Manager web archive (WAR), layouts for the generation of other specific WARs, libraries, the Java API reference documentation, and samples. (Tools for use with Federated Access Manager, including the command line interfaces and policy and authentication agents, can be downloaded separately.) Figure 3–1 illustrates the process for deployment, installation and configuration of a new Federated Access Manager WAR and a patched WAR.

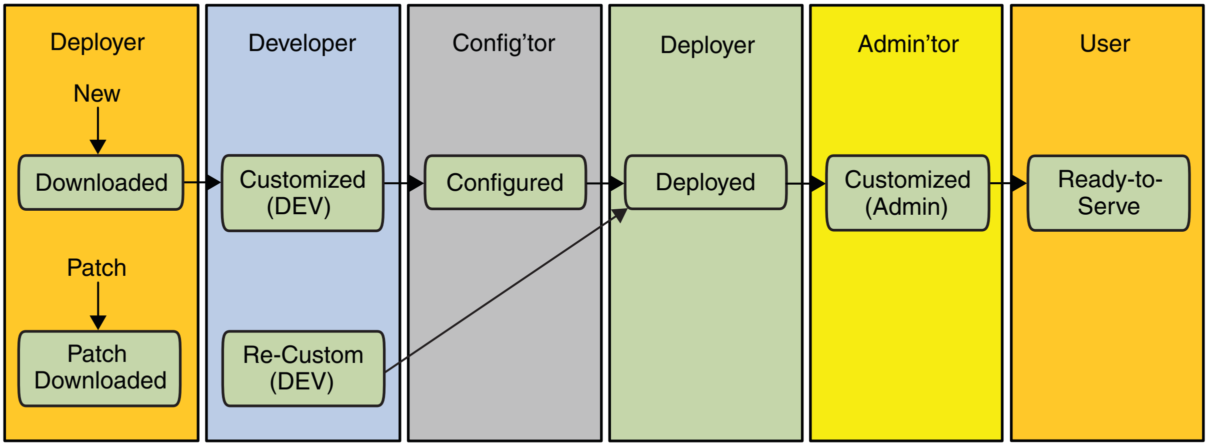

Deployer
New
Downloaded
Patch
Patch Downloaded
Developer
Customized (DEV)
Re-Custom (DEV)
Config'tor
Configured
Deployer
Deployed
Admin'tor
Customized (Admin)
User
Ready-to-Serve

**FIGURE 3–1** Customizing, Patching and Deploying the WAR

---

**Note –** When patched, a full patched version of the Federated Access Manager WAR will be included in the download, assuring that there is always a single download to get the latest bits.

---

The intent of this new process is to allow the administrator to download Federated Access Manager and deploy it on the container or platform of choice, using the web container's administration console or command line interfaces. After the initial launch of the deployed WAR, the user is directed to a JavaServer Page (JSP) called the `Configurator` that prompts for configuration parameters (including, but not limited to, the host name, port number, URI, and repositories), provides data validation for the parameter values to prevent errors, and eliminates post-installation configuration tasks. Once successfully configured, any further changes to the configuration data store must be made using the Federated Access Manager console or command line interfaces.

---

**Note –** When deploying Federated Access Manager against an existing legacy installation, the Directory Management tab will be enabled in the new console.

---

For more information including a directory layout of the ZIP, see the *Sun Federated Access Manager 8.0 Installation and Configuration Guide*.

# Embedded Configuration Data

Federated Access Manager has implemented an embedded configuration data store to replace the `AMConfig.properties` and `serverconfig.xml` files which had been the storage files for server configuration data. Previously, each instance of the server installed had separate configuration files but now when deploying more than one instance of Federated Access Manager, all server configuration data is stored centrally, in one embedded configuration data store per instance. After the Federated Access Manager WAR is configured, a sub configuration is added under the Platform Service to store the data and a bootstrap file that contains the location of the configuration data store is created in the installation directory. Figure 3–2 illustrates how Federated Access Manager is bootstrapped.

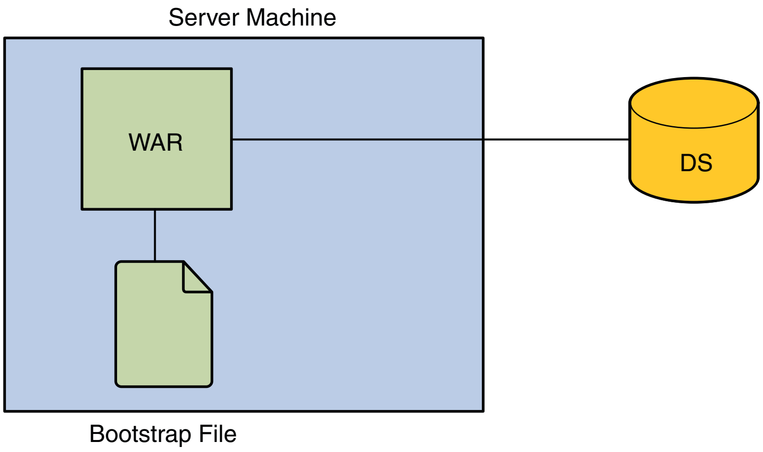


**FIGURE 3–2** Bootstrapping Federated Access Manager

Post-installation, the configuration data can be reviewed and edited using the administration console or the `famadm` command line interface. For more information see the *Sun Federated Access Manager 8.0 Installation and Configuration Guide* and the *Sun Federated Access Manager 8.0 Administration Guide*.

---

**Note –** Federated Access Manager also supports an LDAPv3–based solution that uses an existing directory server for configuration data storage. This is configured during installation. Supported directories include Sun Java System Directory Server, Microsoft Active Directory, and IBM Tivoli Directory.

---

# Centralized Agent Configuration

Policy agents function based on a set of configuration properties. Previously, these properties were stored in the `AMAgent.properties` file, residing on the same machine as the agent. With Centralized Agent Configuration, Federated Access Manager moves most of the agent configuration properties to the embedded configuration data store. Now agent profiles can be configured to store properties locally (on the machine to which the agent was deployed) or centrally (in the embedded configuration data store), making this new function compatible with both older 2.x agents and newer 3.0 agents. Following is an explanation of the local and central agent configuration repositories.

- Local agent configuration is supported for backward compatibility. Agent configuration data is stored in a property file named `FAMAgentConfiguration.properties` that is stored on the agent machine. It is only used by agent profiles configured locally.
- Centralized Agent Configuration stores agent configuration data in a centralized data store managed by Federated Access Manager. When an agent starts up, it reads its bootstrapping file to initialize itself. `FAMAgentBootstrap.properties` is stored on the agent machine and indicates the location from where the configuration properties need to be retrieved. It is used by agent profiles configured locally or centrally. Based on the repository setting in `FAMAgentBootstrap.properties`, it retrieves the rest of its configuration properties. If the repository is local, it reads the agent configuration from a local file; if the repository is remote, it fetches its configuration from Federated Access Manager.

Thus, Centralized Agent Configuration separates the agent configuration properties into two places: a bootstrapping file stored local to the agent and either a local (to the agent) or central (local to Federated Access Manager) agent configuration data store. `FAMAgentBootstrap.properties` is the bootstrapping file used by agent profiles configured locally or centrally. It is stored on the agent machine and indicates the local or central location from where the agent's configuration properties are retrieved. If the repository is local to the agent, it reads the configuration data from a local file; if the repository is remote, it fetches its configuration from Federated Access Manager. Choosing Centralized Agent Configuration provides an agent administrator with the means to manage multiple agent configurations from a central place using either the Federated Access Manager console or command line interface. Figure 3–3 illustrates how an agent retrieves bootstrapping and local configuration data, and configuration data from the configuration data store.

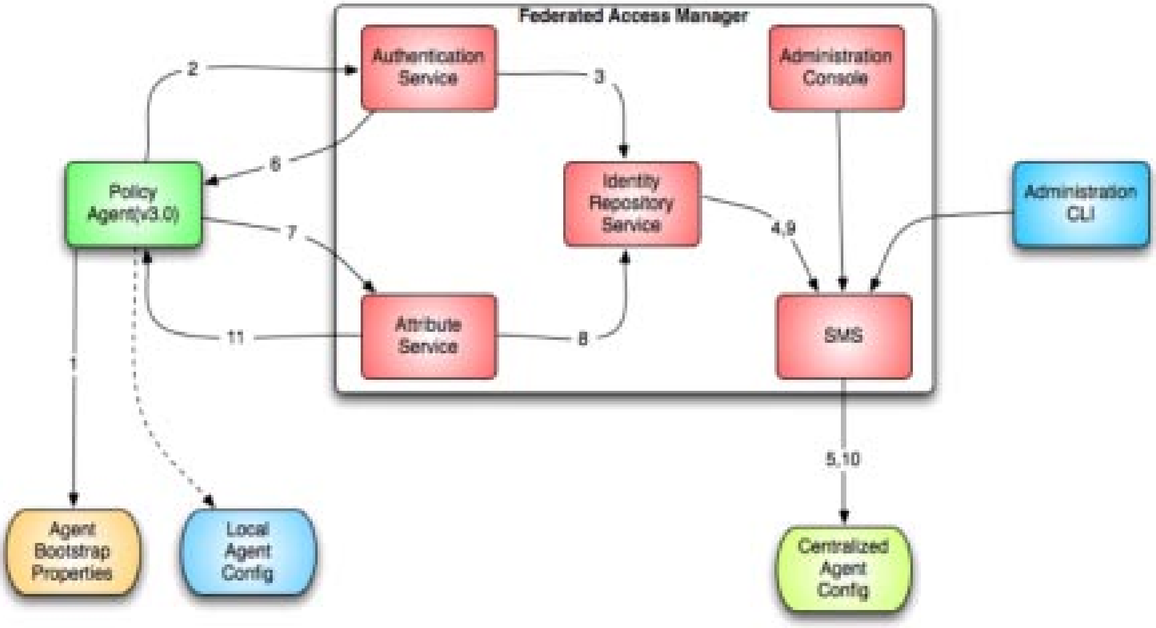

Federated Access Manager
Authentication Service
Administration Console
Policy Agent(v3.0)
Identity Repository Service
Administration CLI
Attribute Service
SMS
Agent Bootstrap Properties
Local Agent Config
Centralized Agent Config
1
2
3
4,9
5,10
6
7
8
11

**FIGURE 3–3** Retrieving Agent Configuration Data

An agent fetches its configuration properties periodically to determine if there have been any configuration changes. Any agent configuration changes made centrally are conveyed to the affected agents which will react accordingly based on the nature of the updated properties. If the properties affected are *hot swappable*, the agent can start using the new values without a restart of the underlying agent web container. Notification of the agent when configuration data changes and polling by the agent for configuration changes can be enabled. Agents can also receive notifications of session and policy changes.

---

**Note –** A agent configuration data change notification does not contain the actual data; it is just a ping that, when received, tells the agent to make a call to Federated Access Manager and reload the latest. Session and policy notifications, on the other hand, contain the actual data changes. Also, when using a load balancer, the notification is sent directly to the agent whose configuration has been changed. It does not go through the load balancer.

---

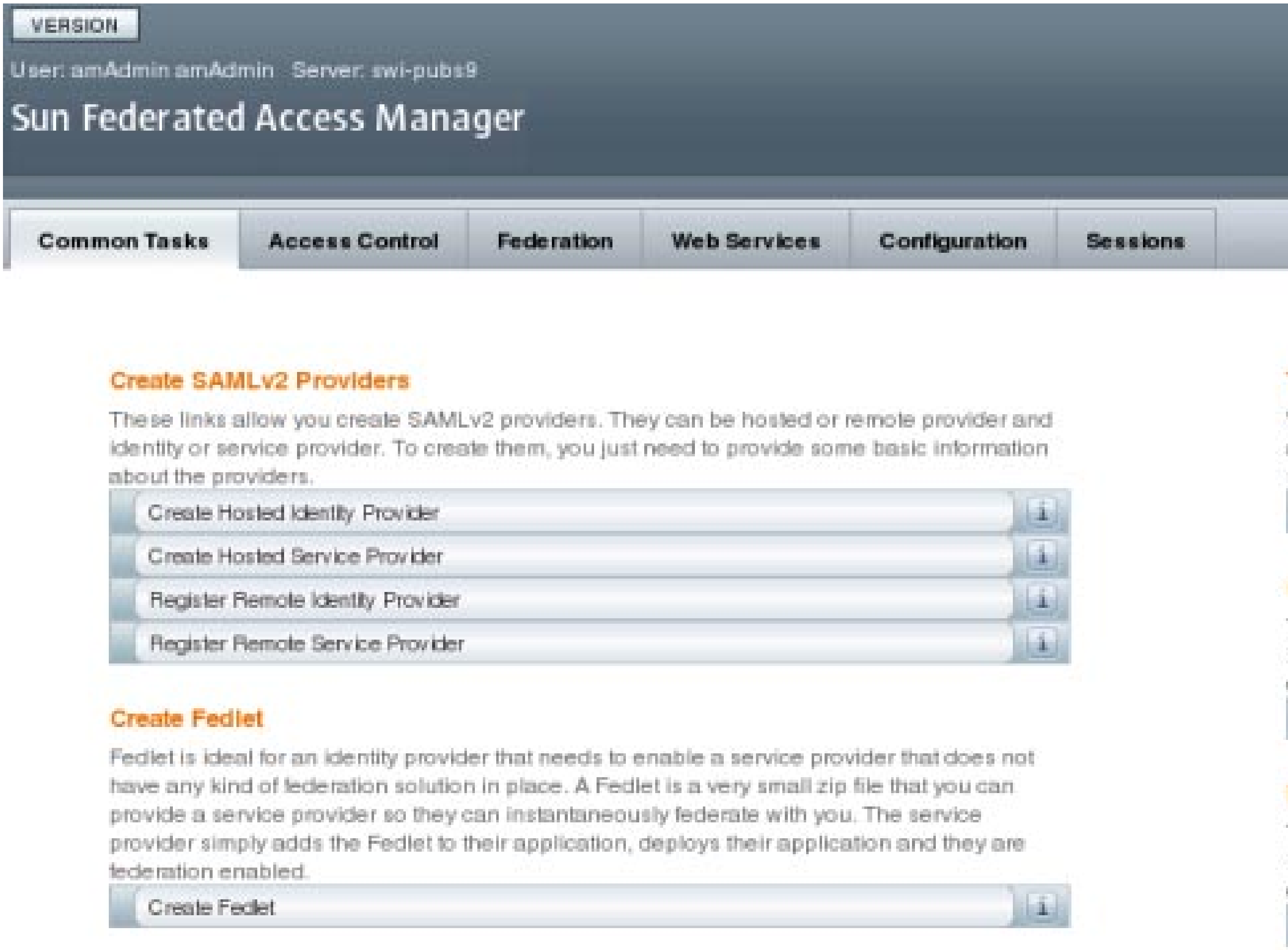
For more information see the *Sun Federated Access Manager 8.0 Administration Guide*.

# Common Tasks

Federated Access Manager has implemented a Common Tasks tab that allows an administrators to create federation-based objects using console wizards. The wizards offer simplified provider configuration with metadata input using the URL `http://fam.sun.com:8080/fam/saml2/jsp/exportmetadata.jsp` or a metadata file. The following things can be done using a Common Task wizard:

- **Create SAML v2 Providers** They can be hosted or remote provider; and identity or service provider. To create them, you just need to provide some basic information about the providers.
- **Create Fedlet** A *Fedlet* is a small ZIP file that can be given to a service provider to allow for immediate federation with an identity provider configured with Federated Access Manager. It is ideal for an identity provider that needs to enable a service provider with no federation solution in place. The service provider simply adds the Fedlet to their application, deploys their application, and they are federation enabled.
- **Test Federation Connectivity** This task validates your federation configuration. It will show if federation connections are being made successfully by identifying where the troubles, if any, are located.
- **Access Documentation** This link opens the OpenSSO documentation page. View frequently asked questions, tips, product documentation, engineering documentation as well as links to the community blogs.
- **Register Your Product** This link allows you to register your product with Sun Connection. You must have a Sun Online Account in order to complete the registration. If you do not already have one, you may request one as part of this process.

Figure 3–4 is a screen capture of the Common Tasks wizard.


VERSION
User: amAdmin amAdmin Server: swi-pubs9
Sun Federated Access Manager
Common Tasks
Access Control
Federation
Web Services
Configuration
Sessions
Create Hosted Identity Provider
Create Hosted Service Provider
Register Remote Identity Provider
Register Remote Service Provider
Create Fedlet

Create SAMLv2 Providers

These links allow you create SAMLv2 providers. They can be hosted or remote provider and identity or service provider. To create them, you just need to provide some basic information about the providers.

Create Fedlet

Fedlet is ideal for an identity provider that needs to enable a service provider that does not have any kind of federation solution in place. A Fedlet is a very small zip file that you can provide a service provider so they can instantaneously federate with you. The service provider simply adds the Fedlet to their application, deploys their application and they are federation enabled.

**FIGURE 3–4** The Common Tasks Wizard

For more information see the *Sun Federated Access Manager 8.0 Administration Guide*.

# Third Party Integration

Federated Access Manager makes it easy to integrate with third-party software. Plug-ins and other tools have been developed to ease the integration of Federated Access Manager and the following products.

- “Sun Java System Identity Manager” on page 72
- “Computer Associates SiteMinder” on page 72

- "Oracle Access Manager" on page 72

For more information, see *Sun Java System Federated Access Manager Integration Guide*.

## Sun Java System Identity Manager

Sun Java System Identity Manager enables an enterprise to manage and audit access to accounts and resources as well as distribute the access management overhead. A Federated Access Manager policy agent is deployed on the Identity Manager machine to regulate access to the Identity Manager server. By mapping Identity Manager objects to Federated Access Manager users and resources, you may significantly increase operational efficiency. For use cases, a technical overview, installation and configuration procedures, architecture diagrams and process flows, see Chapter 1, "Integrating Sun Identity Manager ," in *Sun Java System Federated Access Manager Integration Guide*.

## Computer Associates SiteMinder

Computer Associates SiteMinder (originally developed by Netegrity) is one of the industry's first SSO products — used in a majority of legacy web SSO deployments to protect their intranet and external applications. Federated Access Manager provides the tools for SSO integration with SiteMinder in both intranet and federated environments. They include a SiteMinder Agent and a Federated Access Manager Authentication Module for SiteMinder. They can be found in the `integrations/siteminder` directory of the exploded `fam.war`. For use cases, a technical overview, installation and configuration procedures, architecture diagrams and process flows, see Chapter 2, "Integrating CA SiteMinder," in *Sun Java System Federated Access Manager Integration Guide*.

## Oracle Access Manager

Oracle Access Manager (originally developed by Oblix) is an SSO product with many of the same features as Sun Federated Access Manager and Computer Associates SiteMinder. Oracle Access Manager can be deployed to protect both internal and external applications. Federated Access Manager provides an Oracle Agent and a custom Federated Access Manager Authentication Module for Oracle Access Manager. They can be found in the `integrations/oracle` directory of the exploded `fam.war`. For use cases, a technical overview, installation and configuration procedures, architecture diagrams and process flows, see Chapter 3, "Integrating Oracle Access Manager," in *Sun Java System Federated Access Manager Integration Guide*.

◆ ◆ ◆

## CHAPTER 4

# Deploying Federated Access Manager

Sun Federated Access Manager can be deployed in a number of ways. This chapter contains two sample deployment architectures.

- "Deployment Architecture 1" on page 73
- "Deployment Architecture 2" on page 74

## Deployment Architecture 1

Figure 4–1 illustrates one deployment architecture for Sun Federated Access Manager.

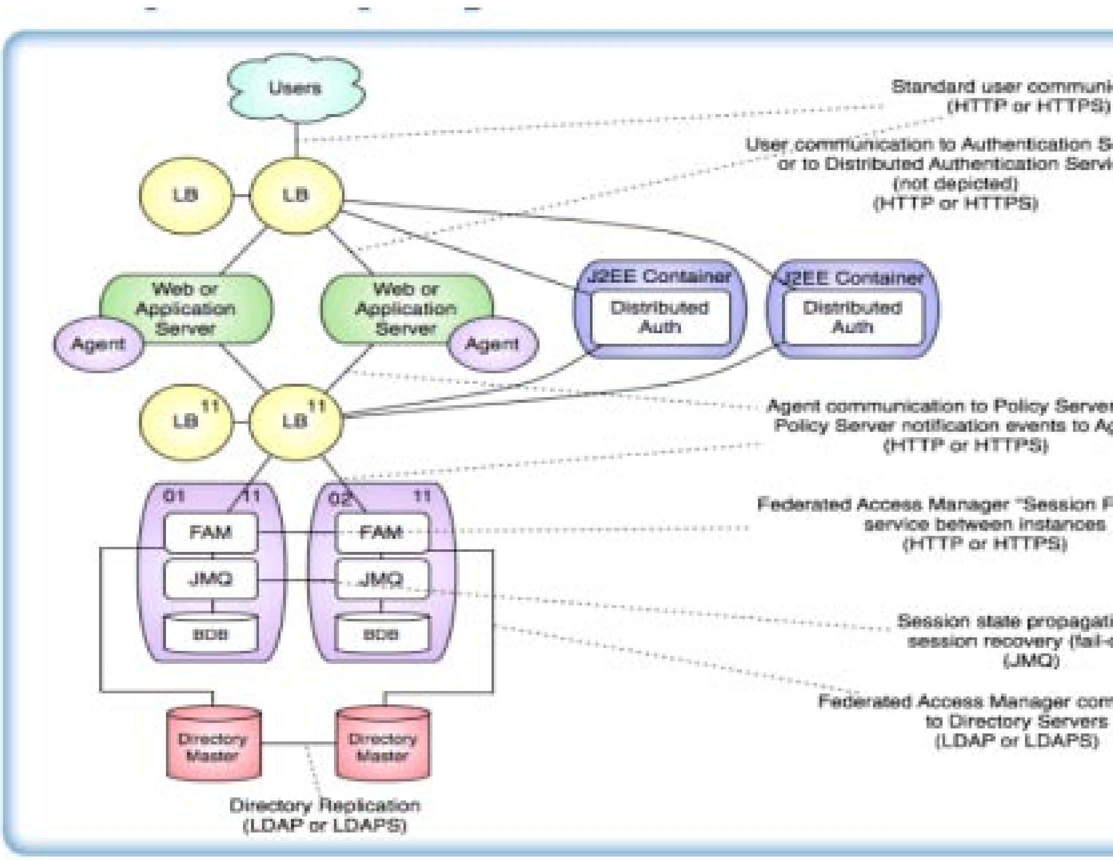

Users
Standard user communi
(HTTP or HTTPS)
User communication to Authentication S
or to Distributed Authentication Servi
(not depicted)
(HTTP or HTTPS)
LB
LB
Web or Application Server
Agent
Web or Application Server
Agent
J2EE Container
Distributed Auth
J2EE Container
Distributed Auth
LB 11
LB 11
Agent communication to Policy Serve
Policy Server notification events to Ag
(HTTP or HTTPS)
01
11
02
11
FAM
FAM
JMQ
JMQ
BDB
BDB
Federated Access Manager "Session F
service between instances
(HTTP or HTTPS)
Session state propagati
session recovery (fail-
(JMQ)
Federated Access Manager com
to Directory Servers
(LDAP or LDAPS)
Directory Master
Directory Master
Directory Replication
(LDAP or LDAPS)

**FIGURE 4–1** Sample Deployment Architecture 1

For more information, see XXXXXXXX. Deployment Planning Guide.

# Deployment Architecture 2

Figure 4–2 illustrates another deployment architecture for Sun Federated Access Manager.

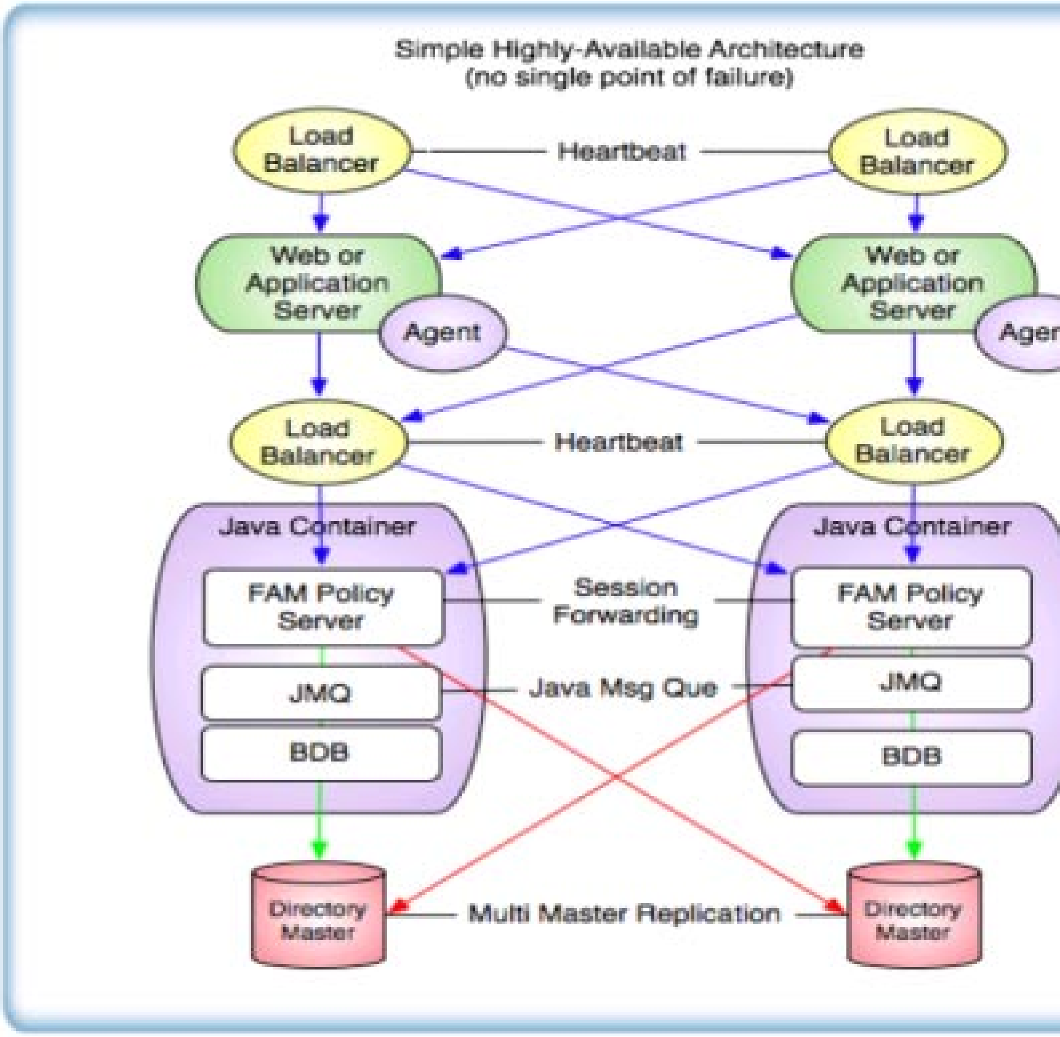

Simple Highly-Available Architecture
(no single point of failure)
Load Balancer
Heartbeat
Load Balancer
Web or Application Server
Agent
Web or Application Server
Ager
Load Balancer
Heartbeat
Load Balancer
Java Container
FAM Policy Server
Session Forwarding
Java Container
FAM Policy Server
JMQ
Java Msg Que
JMQ
BDB
BDB
Directory Master
Multi Master Replication
Directory Master

**FIGURE 4–2** Sample Deployment Architecture 2

For more information, see XXXXXXXX. Deployment Planning Guide.

# PART II

# Access Control Using Federated Access Manager

User authentication, authorization for access to protected resources, and programmatically defining user sessions are all aspects of access management, one of the core functions of Sun Federated Access Manager. Federated Access Manager offers access management features programmatically using the Client SDK, over the wire using HTTP and the Federated Access Manager console, and using an integrated development environment (IDE) application to incorporate Representational State Transfer (REST) calls and Web Services Definition Language (WSDL) files. The chapters in this part contain information on these aspects of access management.

- Chapter 5, “User Sessions and the Session Service”
- Chapter 6, “Models of the User Session and Single Sign-On Processes”
- Chapter 7, “Authentication and the Authentication Service”
- Chapter 8, “Authorization and the Policy Service”

◆ ◆ ◆

## CHAPTER 5

# User Sessions and the Session Service

The Session Service in Sun Federated Access Manager tracks a user's interaction with web applications through the use of session data structures, session tokens, cookies, and other objects. This chapter explains these concepts and other components of a user's session and contains the following sections:

- "About the Session Service" on page 79
- "User Sessions and Single Sign-on" on page 80
- "Session Data Structures and Session Token Identifiers" on page 81

## About the Session Service

The Session Service in Sun Federated Access Manager tracks a user's interaction with protected web applications. For example, the Session Service maintains information about how long a user has been logged in to a protected application, and enforces timeout limits when necessary. Additionally, the Session Service:

- Generates session identifiers.
- Maintains a master copy of session state information.
- Implements time-dependent behavior of sessions.
- Implements session life cycle events such as logout and session destruction.
- Generates session life cycle event notifications.
- Generates session property change notifications.
- Implements session quota constraints.
- Implements session failover.
- Enables single sign-on and cross-domain single sign-on among applications external to Federated Access Manager.
- Offers remote access to the Session Service through the Client SDK with which user sessions can be validated, updated, and destroyed.

The state of a particular session can be changed by user action or timeout. Figure 5–1 illustrates how a session is created as *invalid* before authentication, how it is activated following a successful authentication, and how it can be invalidated (and destroyed) based on timeout values.

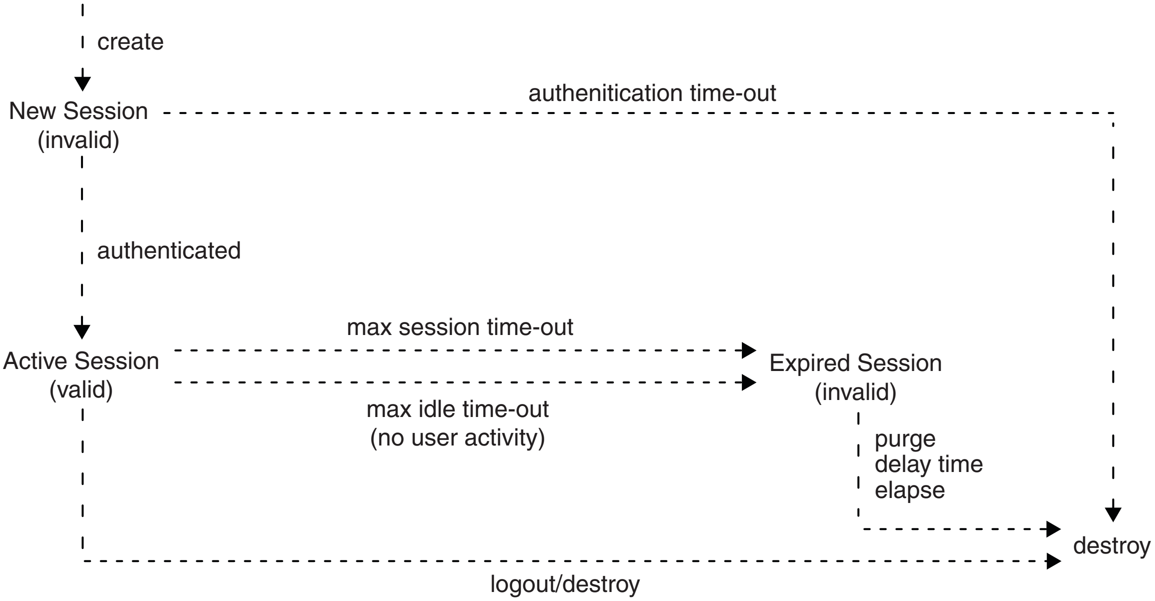

create
New Session
(invalid)
authenitication time-out
authenticated
Active Session
(valid)
max session time-out
max idle time-out
(no user activity)
Expired Session
(invalid)
purge
delay time
elapse
destroy
logout/destroy

**FIGURE 5–1** Life Cycle of a Session

# User Sessions and Single Sign-on

A *user session* is the interval between the moment a user attempts to log in to a resource protected by Federated Access Manager, and the moment the session expires, is terminated by an administrator, or the user logs out. As an example of a user session, an employee attempts to access the corporate benefits administration application protected by Federated Access Manager. A new invalid session is created, and the Authentication Service prompts the user for a username and password to verify the user's identity. Following a successful authentication, the Policy Service and policy agent work together to check that the user has the appropriate permissions to access the protected application and allows or denies access based on the outcome.

Oftentimes, in the same user session (without logging out of the corporate benefits application), the same employee might attempt to access a corporate expense reporting application. Because the expense reporting application is also protected by Federated Access Manager, the Session Service provides proof of the user’s authentication, and the employee is allowed to access the expense reporting application (based on the outcome of a second authorization check with the Policy Service). If access is granted, the employee has accessed more than one application in a single user session without having to reauthenticate. This is called *single sign-on* (SSO). When SSO occurs among applications in more than one DNS domain, it is called *cross-domain single sign-on* (CDSSO). For a more detailed overview of a basic user session, an SSO session, and a CDSSO session, see Chapter 6, “Models of the User Session and Single Sign-On Processes.”

# Session Data Structures and Session Token Identifiers

The Session Service programmatically creates a session data structure to store information about a user session. The result of a successful authentication results in the validation of a *session data structure* for the user or entity and the creation of a *session token* identifier. The session data structure minimally stores the following information.

| | |
|---|---|
| Identifier | A unique, universal identifier for the session data structure. |
| Host Name or IP Address | The location from which the client (browser) is making the request. |
| Principal | Set to the user's distinguished name (DN) or the application's principal name. |
| Type | USER or APPLICATION |
| Session State | Defines whether the session is valid or invalid. |
| Maximum Idle Time | Maximum number of minutes without activity before the session will expire and the user must reauthenticate. |
| Maximum Session Time | Maximum number of minutes (activity or no activity) before the session expires and the user must reauthenticate. |
| Maximum Caching Time | Maximum number of minutes before the client contacts Access Manager to refresh cached session information. |

A session can also contain additional properties which can be used by other applications. For example, a session data structure can store information about a user’s identity, or about a user’s browser preferences. You can configure Federated Access Manager to include the following types of data in a session:

- Protected properties are only modifiable by the server-side modules (primarily the Authentication Service).
- Custom properties are modifiable remotely by any application which possesses the session identifier.

For a detailed summary of information that can be included in a session, see Chapter 10, “Configuring Federated Access Manager Sessions,” in *Sun Federated Access Manager 8.0 Installation and Configuration Guide*.

The session token, also referred to as a *sessionID* and programmatically as an `SSOToken`, is an encrypted, unique string that identifies the session data structure. As the user visits different protected resources using the browser, the session token is propagated to these resources and is used to retrieve the user's credentials. These credentials are then validated by sending a back-end request (using the Client SDK or a policy agent) to Federated Access Manager which then returns an error or the session's prior authentication data. Sessions (and hence the `SSOToken`) are invalidated when a user logs out, the session expires, or a user in an administrative role invalidates it. With Federated Access Manager, a session token is carried in a *cookie*, an information packet generated by a web server and passed to a web browser. (The generation of a cookie for a user by a web server does not guarantee that the user is allowed access to protected resources. The cookie simply points to information in a data store from which an access decision can be made.)

**Note –** Access to some Federated Access Manager services, such as the Policy Service and the Logging Service, require presentation of both the `SSOToken` of the application as well as the `SSOToken` of the user, allowing only designated applications to access these services.

◆ ◆ ◆

## CHAPTER 6

# Models of the User Session and Single Sign-On Processes

This chapter traces events in a basic user session, a single sign-on session (SSO), a cross-domain single sign-on session (CDSSO), and session termination to give you an overview of the features and processes being invoked. It contains the following sections:

- “Basic User Session” on page 83
- “Single Sign-On Session” on page 93
- “Cross-Domain Single Sign-On Session” on page 95
- “Session Termination” on page 97

## Basic User Session

The following sections describe the process behind a basic user session by tracing what happens when a user logs in to a resource protected by Federated Access Manager. In these examples, the server which hosts an application is protected by a policy agent. The Basic User Session includes the following phases:

- “Initial HTTP Request” on page 83
- “User Authentication” on page 85
- “Session Validation” on page 87
- “Policy Evaluation and Enforcement” on page 89
- “Logging the Results” on page 91

### Initial HTTP Request

When a user initiates a user session by using a browser to access and log in to a protected web-based application, the events illustrated in Figure 6–1 occur. The accompanying text describes the model.

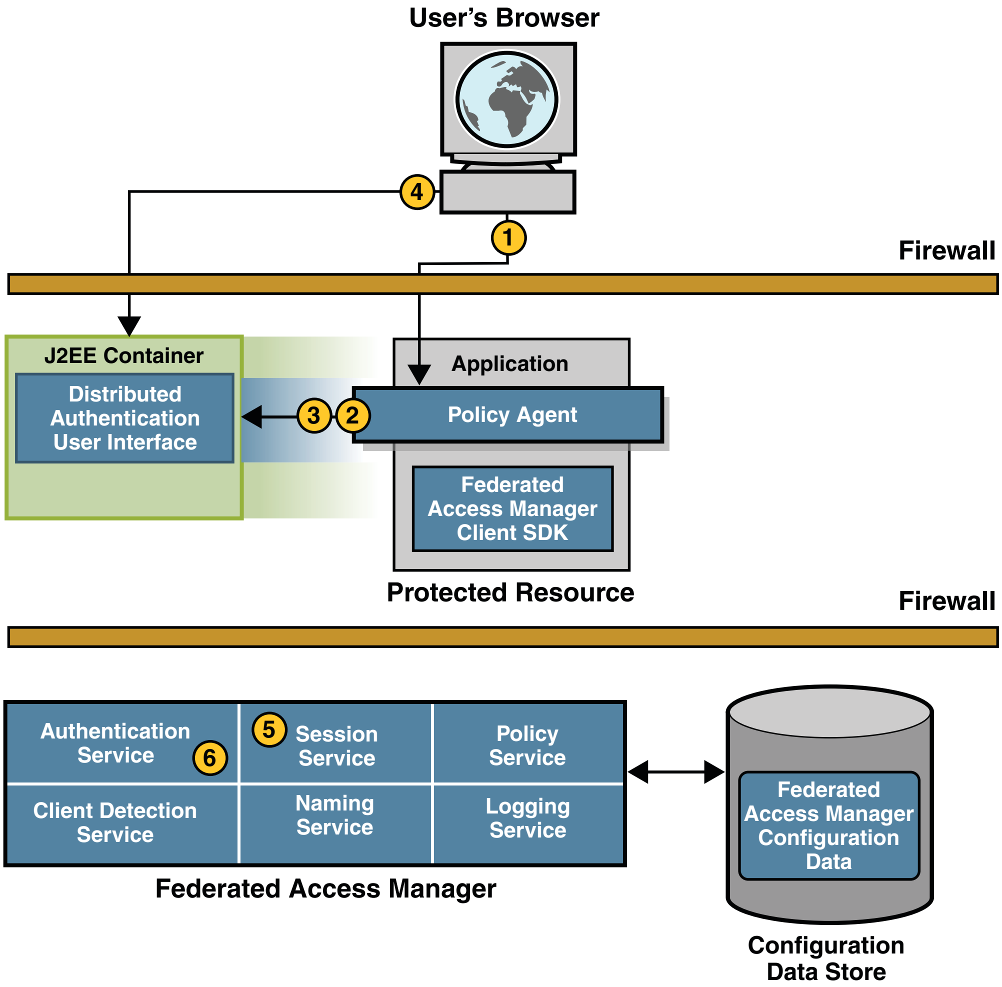

User's Browser
4
1
Firewall
J2EE Container
Distributed Authentication User Interface
3
2
Application
Policy Agent
Federated Access Manager Client SDK
Protected Resource
Firewall
Authentication Service
6
5
Session Service
Policy Service
Client Detection Service
Naming Service
Logging Service
Federated Access Manager
Federated Access Manager Configuration Data
Configuration Data Store

**FIGURE 6–1** Initial HTTP Request

1. The user's browser sends an HTTP request to the protected resource.
2. The policy agent that protects the resource intercepts and inspects the user's request and finds no session token.
3. The policy agent issues a redirect to its configured authentication URL to begin the authentication process.

   In this example, the authentication URL it is set to the URL of the Distributed Authentication User Interface.
4. The browser, following the redirect, sends a GET request for authentication credentials to the Distributed Authentication User Interface.

5. The Session Service creates a new session (session data structure) and generates a session token (a randomly-generated string that identifies the session).
6. The Authentication Service sets the session token in a cookie.

The next part of the user session is "User Authentication" on page 85.

# User Authentication

When the browser sends the GET request to the Distributed Authentication User Interface, the events illustrated in Figure 6–2 occur.

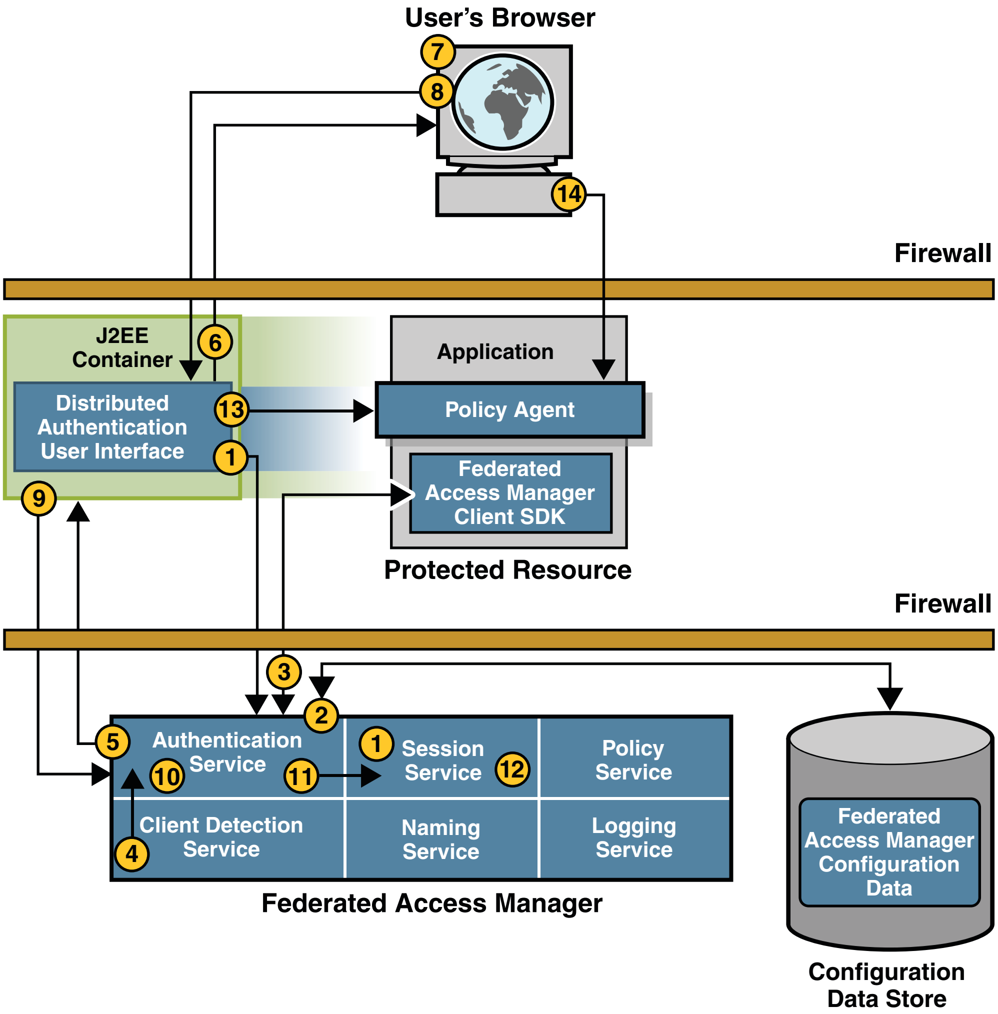

User's Browser
7
8
14
Firewall
J2EE Container
6
Distributed Authentication User Interface
13
1
9
Application
Policy Agent
Federated Access Manager Client SDK
Protected Resource
Firewall
3
2
5
Authentication Service
10
11
1
Session Service
12
Policy Service
Client Detection Service
4
Naming Service
Logging Service
Federated Access Manager
Federated Access Manager Configuration Data
Configuration Data Store

FIGURE 6–2 User Authentication

1. Using the parameters in the GET request, the Distributed Authentication User Interface contacts the Federated Access Manager Authentication Service (which, in turn, communicates with the Session Service).
2. The Authentication Service determines what should be presented to the user based upon configuration data and retrieves the appropriate authentication module(s) and callback(s) information.

   For example, if configured to use LDAP Authentication, the Authentication Service determines that the LDAP Authentication login page should be displayed.

3. The collected information is passed to the Distributed Authentication User Interface using the Client SDK.
4. The Client Detection Service determines which protocol, such as HTML or WML, to use to display the login page.
5. The Distributed Authentication User Interface generates a dynamic presentation extraction page that contains the appropriate credentials request and callbacks information obtained from Federated Access Manager.

   The session cookie will be included in this communication.
6. The user's browser displays the login page.
7. The user enters information in the fields of the login page.
8. The browser sends the credentials in an HTTP POST to the Distributed Authentication User Interface.
9. The Distributed Authentication User Interface uses the Client SDK to pass the credentials to the Authentication Service.
10. The Authentication Service uses the appropriate authentication module to validate the user's credentials.

    For example, if LDAP authentication is used, the LDAP authentication module verifies that the username and password provided exist in the LDAP directory.
11. Assuming authentication is successful, the Authentication Service activates the session by calling the appropriate methods in the Session Service.

    The Authentication Service stores information such as login time, Authentication Scheme, and Authentication Level in the session data structure.
12. Once the session is activated, the Session Service changes the state of the session token to valid.
13. The Distributed Authentication User Interface replies to the protected resource with the validated `SSOToken` in a set-cookie header.
14. Now, the browser makes a second request to the original resource protected by a policy agent.

    This time, the request includes a valid session token created during the authentication process.

The next part of the user session is "Session Validation" on page 87.

## Session Validation

After successful authentication, the user's browser redirects the initial HTTP request to the server a second time for validation. The request now contains a session token in the same DNS domain as Federated Access Manager. The events in Figure 6–3 illustrate this process.

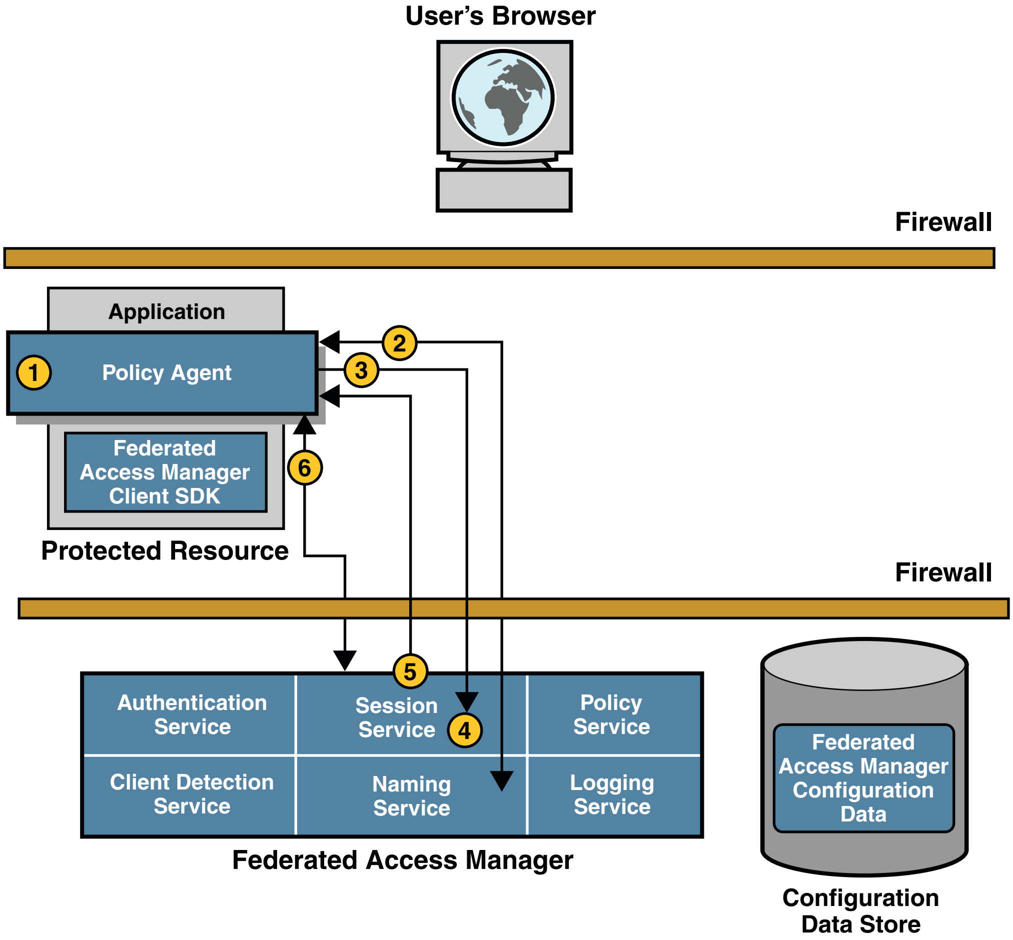

User's Browser
Firewall
Application
1
Policy Agent
2
3
6
Federated Access Manager Client SDK
Protected Resource
Firewall
5
Authentication Service
Session Service
4
Policy Service
Client Detection Service
Naming Service
Logging Service
Federated Access Manager
Federated Access Manager Configuration Data
Configuration Data Store

**FIGURE 6–3** Session Validation

1. The policy agent intercepts the second access request.
2. To determine the validity of the session token, the policy agent contacts the Naming Service to learn where the session token originated.

   The Naming Service allows clients to find the URL for internal Federated Access Manager services. When contacted, the Naming Service decrypts the session token and returns the corresponding URL which can be used by other services to obtain information about the user session.
3. The policy agent, using the information provided by the Naming Service, makes a POST request to the Session Service to validate the included session token.

4. The Session Service receives the request and determines whether the session token is valid based on the following criteria:
    a. Has the user been authenticated?
    b. Does a session data structure associated with the session token exist?
5. If all criteria are met, the Session Service responds that the session token is valid.

    This assertion is coupled with supporting information about the user session itself.
6. The policy agent creates a Session Listener and registers it with the Session Service, enabling notification to be sent to the policy agent when a change in the session token state or validity occurs.

The next part of the user session is "Policy Evaluation and Enforcement" on page 89.

## Policy Evaluation and Enforcement

After a session token has been validated, the policy agent determines if the user can be granted access to the server by evaluating its defined policies. Figure 6–4 illustrates this process.

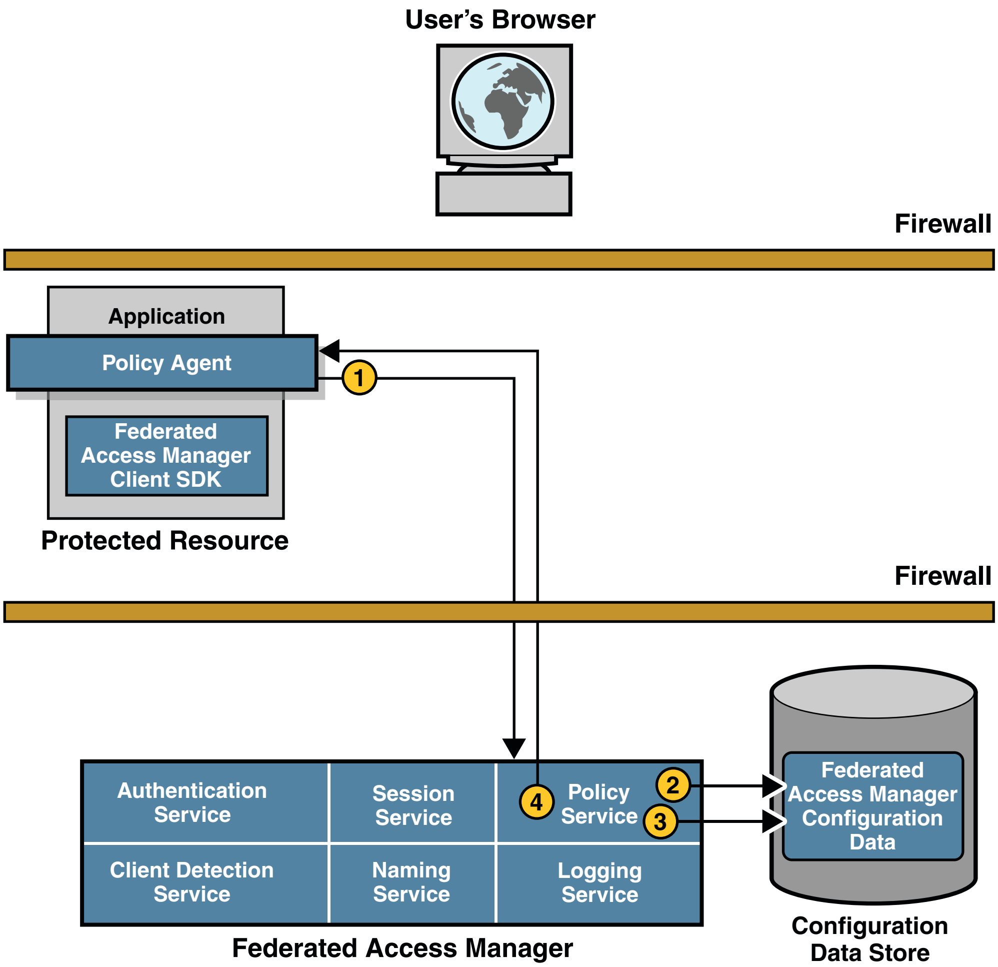

User's Browser
Firewall
Application
Policy Agent
Federated Access Manager Client SDK
Protected Resource
1
Firewall
Authentication Service
Session Service
Policy Service
4
2
3
Client Detection Service
Naming Service
Logging Service
Federated Access Manager
Federated Access Manager Configuration Data
Configuration Data Store

FIGURE 6–4 Policy Evaluation

1. The policy agent sends a request to the Policy Service, asking for decisions regarding resources in its portion of the HTTP namespace.

   The request also includes additional environmental information. For example, IP address or DNS name could be included in the request because they might impact conditions set on a configuration policy.
2. The Policy Service checks for policies that apply to the request.

   Policies are cached in Federated Access Manager. If the policies have not been cached already, they are loaded from Federated Access Manager.
3. If policies that apply to the request are found, the Policy Service checks if the user identified by the session token is a member of any of the Policy Subjects.

   a. If no policies that match the resource are found, the user will be denied access.
   b. If policies are found that match the resource, and the user is a valid subject, the Policy Service evaluates the conditions of each policy. For example, *Is it the right time of day?* or *Are requests coming from the correct network?*
      - If the conditions are met, the policy applies.
      - If the conditions are not met, the policy is skipped.
4. The Policy Service aggregates all policies that apply, encodes a final decision to grant or deny access, and responds to the policy agent.

The next part of the basic user session is "Logging the Results" on page 91.

## Logging the Results

When the policy agent receives a decision from the Policy Service, the events illustrated in Figure 6–5 occur.

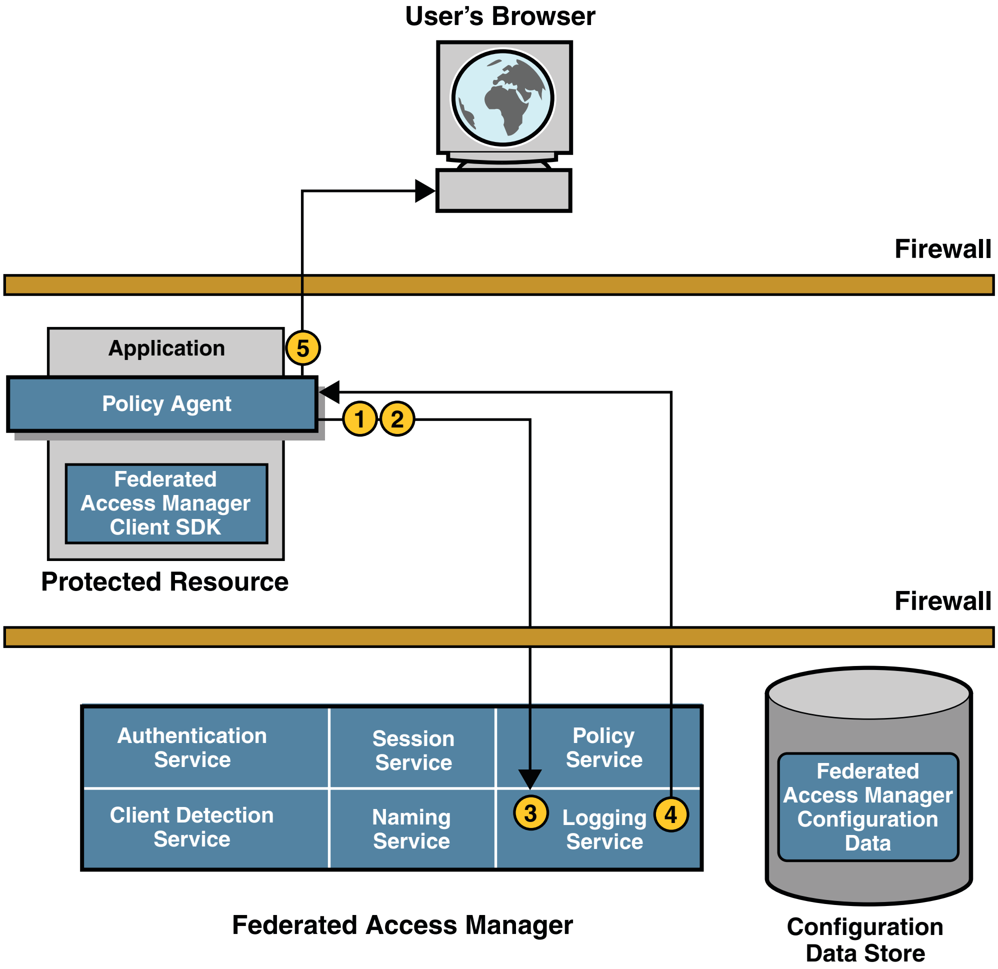

User's Browser
Firewall
Application
5
Policy Agent
1
2
Federated Access Manager Client SDK
Protected Resource
Firewall
Authentication Service
Session Service
Policy Service
Client Detection Service
Naming Service
3
Logging Service
4
Federated Access Manager Configuration Data
Federated Access Manager
Configuration Data Store

FIGURE 6–5 Logging the Policy Evaluation Results

1. The decision and session token are cached by the policy agent so subsequent requests can be checked using the cache (without contacting Federated Access Manager).

   The cache will expire after a (configurable) interval has passed or upon explicit notification of a change in policy or session status.
2. The policy agent issues a logging request to the Logging Service.
3. The Logging Service logs the policy evaluation results to a flat file (which can be signed) or to a JDBC store, depending upon the log configuration.
4. The Logging Service notifies the policy agent of the new log.
5. The policy agent allows or denies the user access to the application.

a. If the user is denied access, the policy agent displays an "access denied" page.
b. If the user is granted access, the resource displays its access page.

Assuming the browser displays the application interface, this basic user session is valid until it is terminated. See "Session Termination" on page 97 for more information. While logged in, if the user attempts to log into another protected resource, the "Single Sign-On Session" on page 93 begins.

## Single Sign-On Session

**Remark 6–1 Reviewer** **There are 13 bullets in graphic and nine steps? Huh????? What should be changed? From email with subject: Auth/SSO process question not answered. check against new graphic**

SSO is always preceded by a basic user session in which a session is created, its session token is validated, the user is authenticated, and access is allowed. SSO begins when the authenticated user requests a protected resource on a second server in the **same** DNS domain. The following process describes an SSO session by tracking what happens when an authenticated user accesses a second application in the same DNS domain as the first application. Because the Session Service maintains user session information with input from all applications participating in an SSO session, in this example, it maintains information from the application the user was granted access to in "Basic User Session" on page 83.

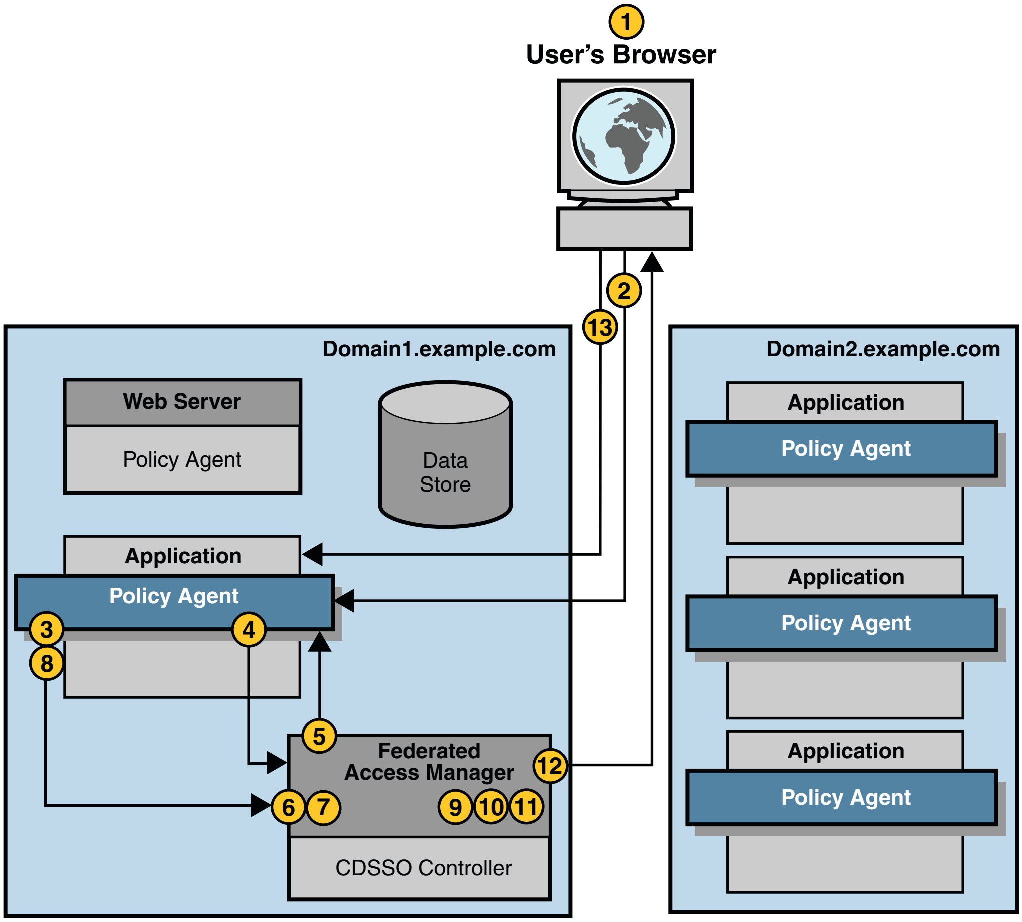

1
User's Browser
2
13
Domain1.example.com
Web Server
Policy Agent
Data Store
Application
Policy Agent
3
8
4
5
Federated Access Manager
12
6
7
9
10
11
CDSSO Controller
Domain2.example.com
Application
Policy Agent
Application
Policy Agent
Application
Policy Agent

**FIGURE 6–6** Single Sign-On Session

1. The user attempts to access a second application hosted on a server in the same domain as the first application to which authentication was successful.
2. The user's browser sends an HTTP request to the second application that includes the user's session token.
3. The policy agent intercepts the request and inspects it to determine whether a session token exists.

   A session token indicates the user is already authenticated. Since the user was authenticated, the Authentication Service is not required at this time. The Session Service API retrieve the session data using the session token identifier imbedded in the cookie.
4. The policy agent sends the session token identifier to the Federated Access Manager Session Service to determine whether the session is valid or not.

For detailed steps, see "Session Validation" on page 87.

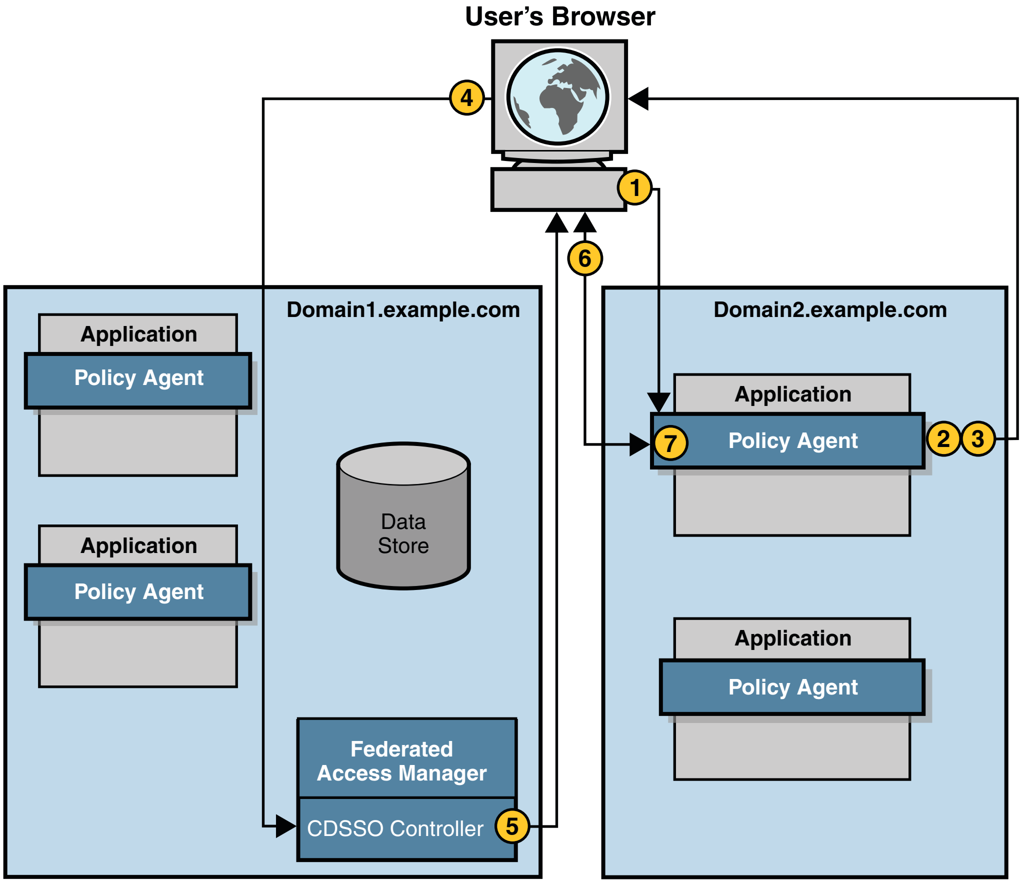
5. The Session Service sends a reply to the policy agent indicating whether the session is valid.
   - If the session is not valid, the user is redirected to the Authentication page.
   - If the session is valid, the Session Service creates a Session Listener.
6. As the session is valid, the Session Service creates a Session Listener.

   A Session Listener sends notification to the policy agent when a change in the session state occurs.
7. The policy agent sends a request for a decision regarding resources in it's portion of the HTTP namespace to the Policy Service.
8. The Policy Service checks for policies that apply to the request.
9. The Policy Service sends the policy evaluation response (either Access Denied or Access Granted.) to the policy agent.
   - If Policy Service does not find policy allowing access to the protected resource, the user is denied access and the Logging Service logs a denial of access. The user may be redirected to a specified page indicating that access was denied if configured as such by the administrator.
   - If the Policy Service finds policy allowing access to the protected resource, the user is granted access and the session is valid until terminated.
10. The policy agent sends a reply to the user indicating whether the user is granted the access.
    - If the user is denied access, the policy agent displays an Access Denied page.
    - If the user is granted access, the protected resource displays its access page.

Assuming the Policy Service finds policy allowing access to the protected resource, the user is granted access and the SSO session is valid until terminated. See "Session Termination" on page 97. While still logged in, if the user attempts to log in to another protected resource located in a different DNS domain, the "Cross-Domain Single Sign-On Session" on page 95 begins.

# Cross-Domain Single Sign-On Session

CDSSO occurs when an authenticated user requests a protected resource on a server in a **different** DNS domain. The user in the previous sections, "Basic User Session" on page 83 and "Single Sign-On Session" on page 93, for example, accessed applications in one DNS domain. In this scenario, the CDSSO Controller within Federated Access Manager transfers the user's session information from the initial domain, making it available to applications in a second domain.


User's Browser
Domain1.example.com
Application
Policy Agent
Application
Policy Agent
Data Store
Federated Access Manager
CDSSO Controller
Domain2.example.com
Application
Policy Agent
Application
Policy Agent
1
2
3
4
5
6
7

1. The authenticated user's browser sends an HTTP request to the application in a different DNS domain.
2. The policy agent intercepts the request and inspects it to determine if a session token exists for the domain *in which the requested application exists*. One of the following occurs:
   - If a session token is present, the policy agent validates the session.
   - If no session token is present, the policy agent (which is configured for CDSSO) will redirect the HTTP request to the CDSSO Controller.

     The CDSSO Controller uses Liberty Alliance Project protocols to transfer sessions so the relevant parameters are included in the redirect.

   In this example, no session token for the second domain is found.
3. The policy agent redirects the HTTP request to the CDSSO Controller.
4. The user's browser allows the redirect to the CDSSO Controller.

Recall that earlier in the user session the session token was set in a cookie in the first domain which is now part of the redirect.

5. The CDC Servlet (in the CDSSO Controller) receives the session token from the first domain, extracts the user's session information, formulates a Liberty POST profile response containing the information, and returns a response to the browser.
6. The user's browser automatically submits the response to the policy agent in the second domain.

   The POST is based upon the Action and the Javascript included in the Body tags `onLoad`.
7. The policy agent in the second domain receives the response, extracts the session information, validates the session, and sets a session token in the cookie for the new DNS domain.

The process continues with "Policy Evaluation and Enforcement" on page 89 and "Logging the Results" on page 91. Based on the policy outcome, the user is granted or denied access to the application.

1. If the user is denied access, the policy agent displays an "access denied" page.
2. If the user is granted access, the protected resource displays its access page. The new cookie can now be used by all agents in the new domain, and the session is valid until it is terminated.

# Session Termination

A user session can be terminated in any of following ways:

- "User Ends Session" on page 97
- "Administrator Ends Session" on page 98
- "Federated Access Manager Enforces Timeout Rules" on page 98
- "Session Quota Constraints" on page 98

## User Ends Session

When a user explicitly logs out of Federated Access Manager by clicking on a link to the Logout Service the following events occur:

1. The Logout Service receives the Logout request, and:
   a. Marks the user's session as destroyed.
   b. Destroys the session.
   c. Returns a successful logout page to the user.
2. The Session Service notifies applications which are configured to interact with the session. In this case, each of the policy agents was configured for Session Notification, and each is sent a document instructing the agent that the session is now invalid.

3. The policy agents flush the session from the cache and the user session ends.

## Administrator Ends Session

Federated Access Manager administrators with appropriate permissions can terminate a user session at any time. When an administrator uses the Sessions tab in the Federated Access Manager console to end a user's session, the following events occur:

1. The Logout Service receives the Logout request, and:
   a. Marks the user's session as destroyed.
   b. Destroys the session.
2. The Session Service notifies applications which are configured to interact with the session. In this case, each of the policy agents was configured for Session Notification, and each is sent a document instructing the agent that the session is now invalid.
3. The policy agents flush the session from cache and the user session ends.

## Federated Access Manager Enforces Timeout Rules

When a session timeout limit is reached, the Session Service:

1. Changes the session status to `invalid`.
2. Displays a time out message to the user.
3. Starts the timer for purge operation delay. (The default is 60 minutes.)
4. Purges or destroys the session when the purge operation delay time is reached.
5. Displays login page to the user if a session validation request comes in after the purge delay time is reached.

## Session Quota Constraints

Federated Access Manager allows administrators to constrain the amount of sessions one user can have. If the user has more sessions than the administrator will allow, one (or more) of the existing sessions can be destroyed.

◆ ◆ ◆

## CHAPTER 7

# Authentication and the Authentication Service

The Sun Federated Access Manager Authentication Service determines whether a user is the person he claims to be. User authentication is the first step in controlling access to web resources within an enterprise. This chapter explains how the Authentication Service works with other components to prove that the user's identity is genuine. Topics covered in this chapter include:

- "Authentication Service Overview" on page 99
- "Authentication Service Features" on page 102
- "Authentication Modules" on page 106
- "Authentication Types" on page 108
- "Configuring for Authentication" on page 110
- "Authentication Graphical User Interfaces" on page 111
- "Authentication Service Programming Interfaces" on page 115

## Authentication Service Overview

The function of the Authentication Service is to request information from an authenticating party, and validate it against the configured identity repository using the specified authentication module. After successful authentication, the user session is activated and can be validated across all web applications participating in an SSO environment. For example, when a user or application attempts to access a protected resource, credentials are requested by one (or more) authentication modules. Gaining access to the resource requires that the user or application be allowed based on the submitted credentials. From the user perspective, a company employee wants to look up a colleague's phone number. The employee uses a browser to access the company's online phone book. To log in to the phone book service, the employee provides a user name and password. Federated Access Manager compares the user's input with data stored in the appropriate identity repository. If Federated Access Manager finds a match for the user name, and if the given password matches the stored password, the user's identity is authenticated.

---

**Note –** The “Basic User Session” on page 83 section in the previous chapter contains a detailed description of the authentication process itself.

---

The Authentication Service can be accessed by a user with a web browser, by an application using the Client SDK, or by any other client that correctly implements the Authentication Service messaging interfaces. The Authentication Service framework has a pluggable architecture for authentication modules that have different user credential requirements. Together with the Session Service, the Authentication Service establishes the fundamental infrastructure for SSO. Generally speaking, the Authentication Service:

- Identifies a requester's credential requirements.
- Generates a dynamic user interface based on the requirements of the authentication module being called.
- Supports custom, pluggable authentication modules.
- Provides pre- and post-processing SPI.
- Populates and manages system domain cookies.
- Generates time dependent alerts and session termination notifications.
- Provides a remote user interface application for distributed deployments.
- Implements a clean logout interface which destroys the session.

Every time a request is used to access the Authentication Service, the session token identifier is retrieved and used to get the associated session data structure from the Session Service. Additionally, the Authentication Service interfaces with the Session Service to:

- Initiate or create user sessions.
- Maintain session state information.
- Activate sessions after successful authentication.
- Populate the valid session data structure with all user-authenticated identity data and properties.
- Destroy sessions after logout.

The following diagram illustrates how the two services work together.

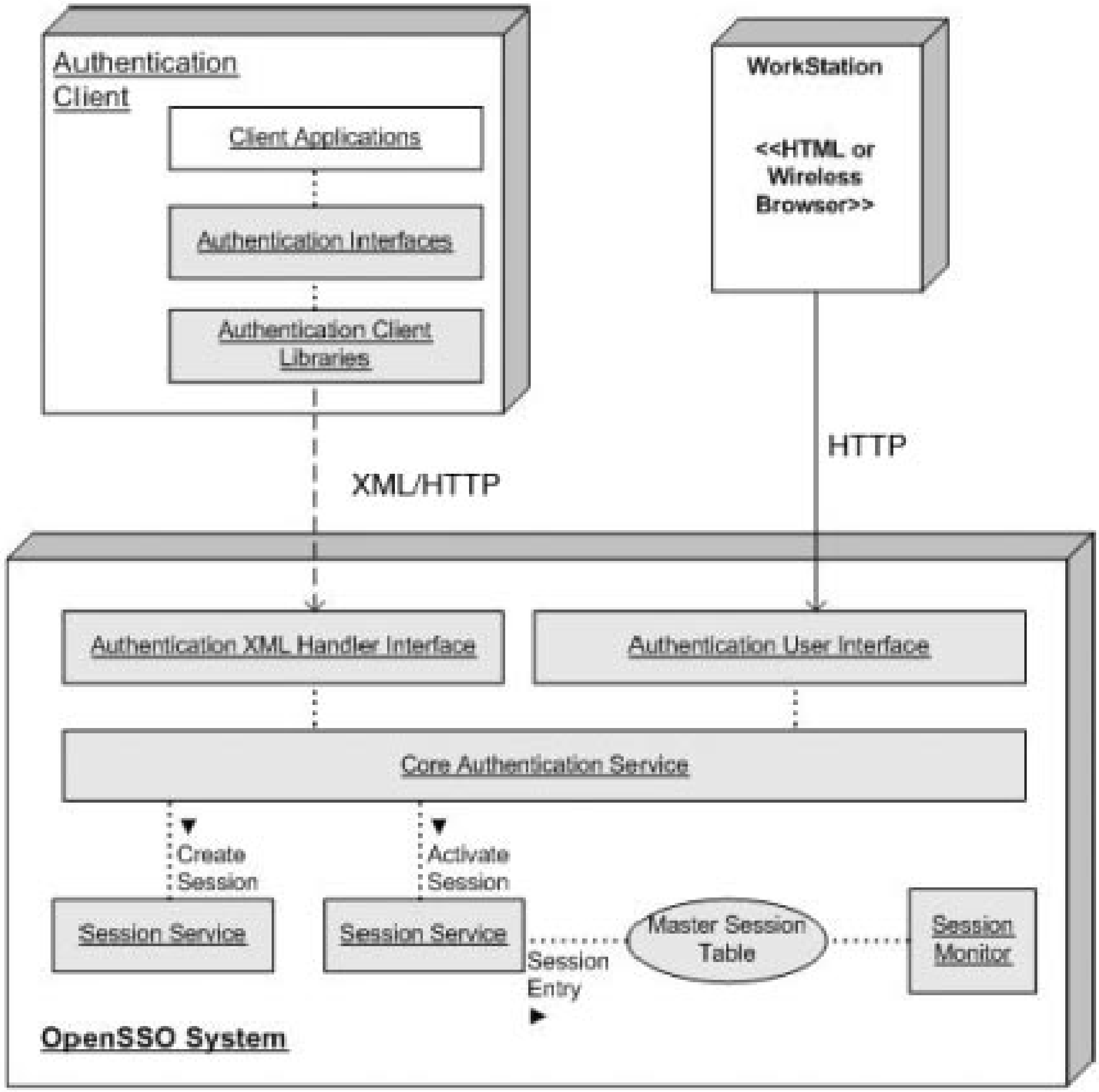

Authentication Client
Client Applications
Authentication Interfaces
Authentication Client Libraries
WorkStation
<<HTML or Wireless Browser>>
XML/HTTP
HTTP
Authentication XML Handler Interface
Authentication User Interface
Core Authentication Service
Create Session
Activate Session
Session Service
Session Service
Session Entry
Master Session Table
Session Monitor
OpenSSO System

**FIGURE 7–1** Authentication Service and Session Service Interfaces

The Authentication Service also interfaces with other Federated Access Manager services including the Naming Service, the Identity Repository Service, the Logging Service, and the Monitoring Service. It also interfaces with the configuration data store and policy agents protecting system resources. (A policy agent must authenticate itself using the Client SDK authentication interfaces, and users with no valid session must be authenticated.)

# Authentication Service Features

The following sections explain some of the features of the Authentication Service.

- "Client Detection" on page 102
- "Account Locking" on page 102
- "Authentication Chaining" on page 103
- "Fully Qualified Domain Name Mapping" on page 104
- "Persistent Cookies" on page 104
- "Session Upgrade" on page 105
- "JAAS Shared State" on page 105
- "Security" on page 105

## Client Detection

Because the Authentication Service is client-type aware, the initial step in the authentication process is to identify the type of client making the HTTP(s) request. This feature is known as *client detection*. The URL information in the HTTP(s) request is used to retrieve the client's characteristics. Based on these characteristics, the appropriate authentication pages are returned. For example, when a web browser is used for requesting access, an HTML login page will be displayed. Once the user is authenticated, the client type is added to the session token for future use. For more information, see Chapter 11, "Identifying the Client Type," in *Sun Federated Access Manager 8.0 Developer's Guide*.

---

**Note –** Federated Access Manager supports all configured client types including cookie-less and cookie-enabled. Currently, there are some restrictions to mobile client detection.

---

## Account Locking

The Authentication Service provides account locking to prevent a user from completing the authentication process after a specified number of failures. Federated Access Manager sends email notifications to administrators when account lockouts occur. Federated Access Manager supports:

| | |
|---|---|
| Physical Locking. | By default, user accounts are physically unlocked. You can initiate physical locking by typing `inactive` as the value of the Lockout Attribute Name attribute in the Core Authentication Service. Additionally, the value of the Login Failure Lockout Duration attribute should be set to `0`.<br>physical lock attr name is: inetuserstatus value active/inactive |
| Memory Locking. | You can configure Memory Locking so that a user account is locked in memory after a specified number of authentication attempts. By |

changing the Login Failure Lockout Duration attribute to a value greater then `0`, the user's account is then locked in memory for the number of minutes specified and the account is unlocked after the time period elapses.

To figure out the amount of time the lockout will be in effect, the value of the Lockout Duration Multiplier attribute is multiplied by the value of the Login Failure Lockout Duration attribute for subsequent lockout. For example, if the value of Login Failure Lockout Duration is 5 minutes and the value of the Lockout Duration Multiplier is 2, the first time a user is locked out in memory will be 5 minutes. The second time this same user gets locked out in memory the lockout duration will be 10 minutes (5 minutes x 2). The third time this same user gets locked out in memory the lockout duration will be 20 minutes (5 minutes x 2 x 2).

The account locking feature is disabled by default. Account locking activities are also logged. For information on how to enable it, see "Account Locking" in *Sun Federated Access Manager 8.0 Administration Guide*.

---

**Note –** Only authentication modules that throw an `Invalid Password Exception` can leverage the Account Locking feature. Out of the box, these include Active Directory, Data Store, HTTP Basic, LDAP, and Membership.

---

# Authentication Chaining

Federated Access Manager allows the configuration of an authentication process in which a user must pass credentials to one or more authentication modules before session validation is accomplished. This is called *authentication chaining*. Federated Access Manager uses the Java Authentication and Authorization Service (JAAS) framework (integrated with the Authentication Service) to implement authentication chaining. The JAAS framework validates all user identifiers used during the authentication process, and maps them to one principal. (The mapping is based on the configuration of the User Alias List attribute in the user's profile.) If all the maps are correct, the session token is validated. If all the maps are not correct, the user is denied a valid session token. Once authentication to all modules in the chain succeeds or fails, control is returned to the Authentication Service from the JAAS framework.

You configure an authentication chain by realm, user, role, or service. Determining validation is based upon the control flag configured for each authentication module instance defined in the chain. The flags are:

| | |
|---|---|
| Requisite | Authentication to this module instance is required to succeed. If it succeeds, authentication continues down the module instance list. If it fails, control immediately returns to the application. |

| | |
|---|---|
| Required | Authentication to this module instance is required to succeed. If any of the required module instances defined in the chain fails, the whole authentication chain will fail. |
| Sufficient | The module instance is not required to succeed. If it does succeed, control immediately returns to the application (authentication does not proceed down the module instance list). If it fails, authentication continues down the list. |
| Optional | The module instance is not required to succeed. Whether it succeeds or fails, authentication still continues to proceed down the module instance list. |

---

**Note –** Role based authentication is only supported for use with the AM SDK data store. This data store would come from an existing Sun Java System Access Manager 7.x installation or would have been manually created.

---

For more information, see "Authentication Modules and Chains" in *Sun Federated Access Manager 8.0 Administration Guide*. For an overview of the authentication module instances, see "Authentication Modules" on page 106.

## Fully Qualified Domain Name Mapping

Fully Qualified Domain Name (FQDN) mapping enables the Authentication Service to take corrective action in the case where a user may have typed in an incorrect URL. This is necessary, for example, when a user specifies a partial host name or IP address to access protected resources. This feature can also be used to allow access to one instance of Federated Access Manager using many different aliases. For example, you might configure one instance of Federated Access Manager as `intranet.example.com` for employees and `extranet.example.com` for partners. For more information, see "Fully Qualified Domain Name Mapping" in *Sun Federated Access Manager 8.0 Administration Guide*.

## Persistent Cookies

A *persistent cookie* is an information packet that is written to the user's hard drive and, therefore, continues to exist after the web browser is closed. The persistent cookie enables a user to log into a new browser session without having to reauthenticate. The Authentication Service can be enabled to write persistent cookies rather than cookies that are written to a web browser's memory. For more information, see "Persistent Cookie" in *Sun Federated Access Manager 8.0 Administration Guide*.

## Session Upgrade

The Authentication Service allows for the upgrade of a valid session based on a second, successful authentication performed by the same user. If a user with a valid session attempts to authenticate to a second resource secured under the realm to which he is currently authenticated, and this second authentication request is successful, the Authentication Service updates the session with the new properties based on the new authentication. If the authentication fails, the current user session is returned without an upgrade. If the user with a valid session attempts to authenticate to a resource secured in a different realm, the user will receive a message asking whether the user would like to authenticate to the new realm. The user can choose to maintain the current session, or can attempt to authenticate to the new realm. Successful authentication will result in the old session being destroyed and a new one being created. For more information, see "Session Upgrade" in *Sun Federated Access Manager 8.0 Administration Guide*.

---

**Note –** Successful authentication for session upgrade does not necessarily destroy the previous session. If the subsequent `AuthContext` object is created with the constructor `AuthContext(SSOToken ssoToken, boolean forceAuth)` when `forceAuth` is set to `true`, the existing session will be used and a new session will not be created.

---

## JAAS Shared State

The JAAS shared state enables sharing of both a user identifier and a password between authentication module instances. Options are defined for each authentication module type by realm, user, service and role. If an authentication fails with the credentials from the shared state, the authentication module restarts the authentication process by prompting for its required credentials. If it fails again, the module is marked failed. After a commit, an abort, or a logout, the shared state will be cleared. For more information, see "JAAS Shared State" in *Sun Federated Access Manager 8.0 Administration Guide*.

## Security

From a security point of view, here are some general practices implemented in the Authentication Service.

- SSL is strongly recommended to prevent the user credentials from being stolen through passive network snooping.
- The signing and encryption of some user data is to prevent other software applications, sharing the same system resources, from subverting it.

- The main user entry points of the Authentication Service (Distributed Authentication User Interface, Authentication XML Handler Interface for remote clients, the Authentication Service User Interface) are protected by entry level validation of the size of the requested data.
- Creation and modification of authentication configuration information is only allowed by privileged Federated Access Manager administrators.

# Authentication Modules

An *authentication module* is a plug-in that collects user information such as a user ID and password, and compares the information against entries in a database. If a user provides information that meets the authentication criteria, the user is validated and, assuming the appropriate policy configuration, granted access to the requested resource. If the user provides information that does not meet the authentication criteria, the user is not validated and denied access to the requested resource. Federated Access Manager is deployed with a number of authentication modules. Table 7–1 provides a brief description of each.

**TABLE 7–1** Authentication Service Modules

| Authentication Module Name | Description |
|---|---|
| Active Directory | Uses an Active Directory operation to associate a user identifier and password with a particular Active Directory entry. You can define multiple Active Directory authentication configurations for a realm. Allows both LDAP and Active Directory to coexist under the same realm. |
| Anonymous | Enables a user to log in without specifying credentials. You can create an Anonymous user so that anyone can log in as Anonymous without having to provide a password. Anonymous connections are usually customized by the Federated Access Manager administrator so that Anonymous users have limited access to the server. |
| Certificate | Enables a user to log in through a personal digital certificate (PDC). The user is granted or denied access to a resource based on whether or not the certificate is valid. The module can optionally require the use of the Online Certificate Status Protocol (OCSP) to determine the state of a certificate. |
| Data Store | Enables authentication against one or more configuration data stores within a realm. |

**TABLE 7–1** Authentication Service Modules *(Continued)*

| Authentication Module Name | Description |
|---|---|
| Federation | Used by the service provider during federation (using SAML v1.x, SAML v2, WS-Federation, Liberty ID-FF) to create a session after validating the assertion. This authentication module can not be invoked like the other modules as it is invoked directly by the `SAMLAwareServlet`. |
| HTTP Basic | Enables authentication to occur with no data encryption. Credentials are validated internally using either the LDAP or Data Store authentication module. |
| Java Database Connectivity (JDBC) | Enables authentication through any Structured Query Language (SQL) databases that provide JDBC-enabled drivers. The SQL database connects either directly through a JDBC driver or through a JNDI connection pool. |
| LDAP | Enables authentication using LDAP bind, a directory server operation which associates a user identifier and password with a particular LDAP entry. You can define multiple LDAP authentication configurations for a realm. |
| Membership | Enables user to self-register a user entry. The user creates an account, personalizes it, and accesses it as a registered user without the help of an administrator. Implemented similarly to personalized sites such as `my.site.com`or `mysun.sun.com`. |
| MSISDN | The Mobile Station Integrated Services Digital Network (MSISDN) authentication module enables authentication using a mobile subscriber ISDN associated with a device such as a cellular telephone. It is a non-interactive module. The module retrieves the subscriber ISDN and validates it against the user repository to find a user that matches the number. |
| RADIUS | Uses an external Remote Authentication Dial-In User Service (RADIUS) server to verify identities. |
| Security Assertion Markup Language (SAML) | Receives and validates SAML assertions on a target server by using either a web artifact or a POST response. |
| SafeWord® | Uses Secure Computing's SafeWord PremierAccess™ server software and SafeWord tokens to verify identities. |
| SecurID™ | Uses RSA ACE/Server software and RSA SecurID authenticators to verify identities. |
| UNIX® | Solaris and Linux modules use a user's UNIX identification and password to verify identities. |

**TABLE 7–1** Authentication Service Modules *(Continued)*

| Authentication Module Name | Description |
|---|---|
| Windows Desktop Single Sign-On (SSO) | Allows a user who has already authenticated with a key distribution center to be authenticated by Federated Access Manager without having to provide the login information again. Leverages Kerberos authentication and is supported wherever Kerberos is supported (including Windows, Solaris, Linux, and Macintosh). |
| Windows NT | Uses a Microsoft Windows NT™ server to verify identities. |

You can use the Federated Access Manager console to enable and configure the authentication modules. You can also create and configure multiple instances of a particular authentication module. (An *authentication module instance* is a child entity that extends the schema of a parent authentication module and adds its own subschema.) Finally, you can write your own custom authentication module (or plug-in) to connect to the Federated Access Manager authentication framework. See Chapter 4, "Managing Authentication," in *Sun Federated Access Manager 8.0 Administration Guide* for detailed information about enabling and configuring default authentication modules and authentication module instances. See Chapter 2, "Using the Authentication Interfaces," in *Sun Federated Access Manager 8.0 Developer's Guide* for more information about writing custom authentication modules.

# Authentication Types

After granting or denying access to a resource, Federated Access Manager checks for information about where to redirect the user. A specific order of precedence is used when checking for this information. The order is based on whether the user was granted or denied access to the protected resource, and on the type of authentication specified. When you install Federated Access Manager, a number of authentication types are automatically configured.

| | |
|---|---|
| Realm-based Authentication. | User authenticates to a configured realm or sub-realm.<br>**Note –** This authentication type is equivalent to organization—based authentication. The query parameters `org` and `realm` would both lead to realm-based authentication in realm mode, and organization-based authentication in legacy mode. |
| Role-based Authentication. | User authenticates to a configured role within a realm or sub-realm. The user must possess the |

| | |
|---|---|
| | role. A *static role* is possessed when an attribute is assigned to a specific user or container. A *filtered role* is dynamically generated based on an attribute contained in the user's or container's entry. For example, all users that contain a value for the `employee` attribute can be included in a role named *employees* when the filtered role is created.<br>**Note –** Role based authentication is only supported for use with the AM SDK data store. This data store would come from an existing Sun Java System Access Manager 7.x installation or would have been manually created. |
| Service-based Authentication. | User authenticates to a specific service or application registered to a realm or sub-realm. |
| User-based Authentication. | User authenticates using an authentication process configured specifically for him or her. |
| Authentication Level-based Authentication | An administrator specifies the security level of the authentication modules by defining each with an *authentication level*. Successful authentication to a higher authentication level defines a higher level of trust for the user. If a user attempts to access a service, the service can determine if the user is allowed access by checking the authentication level in the user's session data. If the authentication level is not high enough, the service redirects the user to go through an authentication process with a set authentication level. |
| Module-based Authentication. | Allows a user to specify the module to which they will authenticate. |
| Organization-based Authentication. | User authenticates to an organization or sub-organization. |

---

**Note –** This authentication type is equivalent to realm-based authentication. The query parameters `org` and `realm` would both lead to realm-based authentication in realm mode, and organization-based authentication in legacy mode.

---

For more information, see "Authentication Types" in *Sun Federated Access Manager 8.0 Administration Guide*.

# Configuring for Authentication

The authentication framework includes the following places where you can configure for authentication:

- "Core Authentication Module and Realm Configuration" on page 110
- "Authentication Configuration Service" on page 111
- "Login URLs and Redirection URLs" on page 111

Explanations of the authentication attributes can be found in the Online Help and the Part II, "Configuration Attribute Reference," in *Sun Federated Access Manager Administration Reference*.

## Core Authentication Module and Realm Configuration

The Core Authentication Module contains general authentication properties that can be defined globally using the Federated Access Manager console (under the Configuration tab) or more specifically for each configured realm (under the Access Control tab). Core authentication properties are added and enabled for the top-level realm during installation. As new realms are configured under the top-level realm, these properties (and the values defined globally for them) are dynamically added to each new realm when it is created. Once added, new values can be defined and configured values can be modified by the realm's administrator. The values are then used if no overriding value is defined in the specified authentication module instance or authentication chain. The default values for the Core Authentication Module are defined in the `amAuth.xml` file and stored in the configuration data store. For more information, see "General Authentication Properties" in *Sun Federated Access Manager 8.0 Administration Guide* and the Part II, "Configuration Attribute Reference," in *Sun Federated Access Manager Administration Reference*.

## Authentication Configuration Service

The Authentication Configuration Service describes all the dynamic attributes for service-based authentication. This service is used for configuring roles. When you assign a service to a role, you can also assign other attributes such as a success URL or an authentication post-processing class to the role. For more information, see "Role-based Authentication" in *Sun Federated Access Manager 8.0 Administration Guide*.

## Login URLs and Redirection URLs

In the last phase of the authentication process, Federated Access Manager either grants or denies access to the user. If access is granted, Federated Access Manager uses a login URL to display a page in the browser. If access is denied, Federated Access Manager uses a redirection URL to display an alternate page in the browser. A typical alternate page contains a brief message indicating the user has been denied access.

Each authentication type (as discussed in "Authentication Types" on page 108) uses a login URL or redirection URL based on a specific order of precedence, and on whether the authentication succeeded or failed. For a detailed description of how Federated Access Manager proceeds through the order of precedence, see "Authentication Types" in *Sun Federated Access Manager 8.0 Administration Guide*.

# Authentication Graphical User Interfaces

The Federated Access Manager Authentication Service has two separate graphical user interfaces that can be used. The following sections contain information on them.

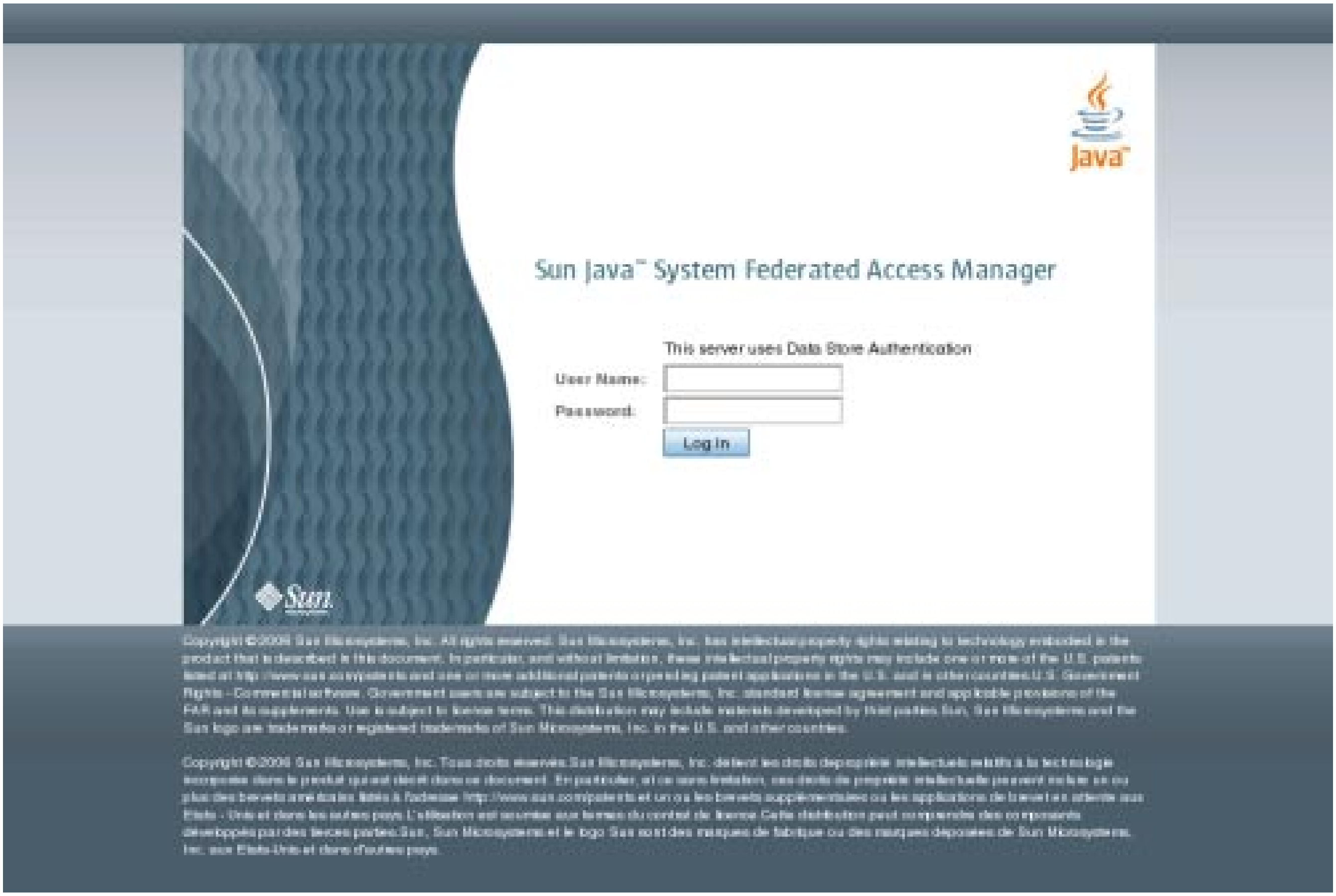
- "Authentication Service User Interface" on page 111
- "Distributed Authentication User Interface" on page 113

## Authentication Service User Interface

The Authentication Service implements a user interface that is separate from the Federated Access Manager administration console. The Authentication Service user interface provides a dynamic and customizable means for gathering authentication credentials. When a user requests access to a protected resource, the Authentication Service presents a web-based login page and prompts the user for the appropriate credentials based on the configured authentication module or chain. Once the credentials have been passed back to Federated Access Manager and authentication is deemed successful, the user may gain access to the protected resource if authorized to do so. The Authentication Service user interface can be used for the following:

- Administrators can access the administration portion of the Federated Access Manager console to manage their realm's identity data.
- Users can access their own profiles to modify personal data.
- A user can access a resource defined as a redirection URL parameter appended to the login URL.
- A user can access the resource protected by a policy agent.

Below is a screen capture of the default Authentication Service user interface.



This server uses Data Store Authentication

FIGURE 7–2 Authentication Service User Interface

Federated Access Manager provides customization support for the Authentication Service user interface. You can customize JavaServer Pages™ (JSP™) and the file directory level by organization, service, locale, or client type. See Chapter 16, “Customizing the Authentication User Interface,” in *Sun Federated Access Manager 8.0 Developer’s Guide* for more information.

## Distributed Authentication User Interface

Federated Access Manager also provides a remote authentication user interface component to enable secure, distributed authentication across two firewalls. A web browser communicates an HTTP request to the remote authentication user interface which, in turn, presents the appropriate module login page to the user. The web browser then sends the user login information through a firewall to the remote authentication user interface which, in turn, communicates through the second firewall with Federated Access Manager. The Distributed Authentication User Interface enables a policy agent or an application that is deployed in a non-secured area to communicate with the Federated Access Manager Authentication Service installed in a secured area of the deployment. Figure 7–3 illustrates this scenario.

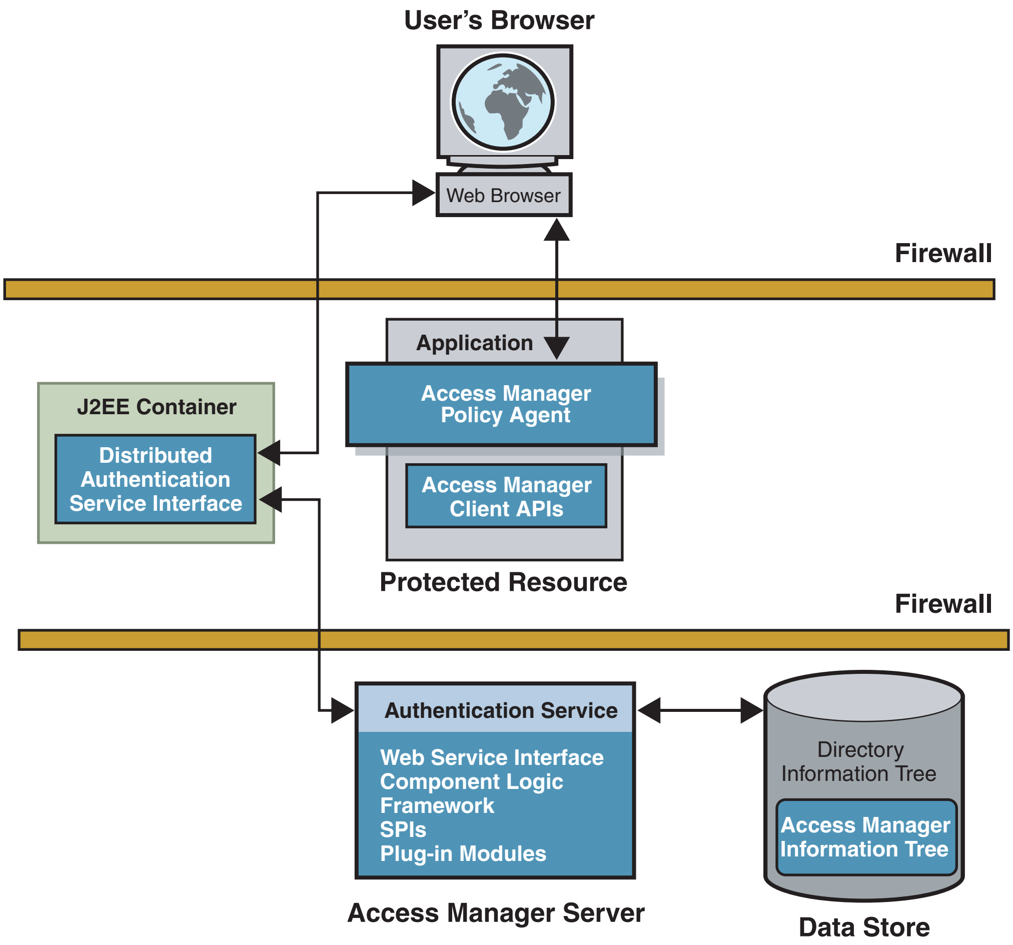

User's Browser
Web Browser
Firewall
Application
Access Manager Policy Agent
J2EE Container
Distributed Authentication Service Interface
Access Manager Client APIs
Protected Resource
Firewall
Authentication Service
Web Service Interface
Component Logic
Framework
SPIs
Plug-in Modules
Access Manager Server
Directory Information Tree
Access Manager Information Tree
Data Store

FIGURE 7–3 Distributed Authentication Process

The Distributed Authentication User Interface uses a JATO presentation framework and is customizable. (See screen capture in "Authentication Service User Interface" on page 111.) You can install the Distributed Authentication User Interface on any servlet-compliant web container within the non-secure layer of a Federated Access Manager deployment. The remote component then works with the Authentication client APIs and authentication utility classes to authenticate web users. For a more detailed process, see "User Authentication" on page 85. For detailed installation and configuration instructions, see Chapter 6, "Deploying a Distributed Authentication UI Server," in *Sun Federated Access Manager 8.0 Installation and Configuration Guide*.

# Authentication Service Programming Interfaces

Federated Access Manager provides both Java APIs and C APIs for writing authentication clients that remote applications can use to gain access to the Authenticate Service. Communication between the APIs and the Authentication Service occurs by sending XML messages over HTTP(S). The Java and C APIs support all authentication types supported by the browser-based user interface. Clients other than Java and C clients can use the XML/HTTP interface directly to initiate an authentication request. Additionally, you can add custom authentication modules to Federated Access Manager by using the service provider interface (SPI) package, `com.iplanet.authentication.spi`. This SPI implements the JAAS `LoginModule`, and provides additional methods to access the Authentication Service and module configuration properties files. Because of this architecture, any custom JAAS authentication module will work within the Authentication Service. For more information, see Chapter 2, “Using the Authentication Interfaces,” in *Sun Federated Access Manager 8.0 Developer’s Guide* and *Federated Access Manager 8.0 Java API Reference*.

Federated Access Manager also provides a Client SDK that can implement authentication logic on a remote web server or application server. For information, see Chapter 1, “Enhancing Remote Applications Using the Client Software Development Kit,” in *Sun Federated Access Manager 8.0 Developer’s Guide*.

◆ ◆ ◆ CHAPTER 8

# Authorization and the Policy Service

The Sun Federated Access Manager Policy Service determines if a user has been given permission by a recognized authority to access a protected resource. The process is referred to as *authorization*. This chapter describes how the various parts of the Policy Service work together to perform authorization. Topics covered include:

- "Authorization and Policy Service Overview" on page 117
- "Policy Types" on page 119
- "Realms and Access Control" on page 122
- "Policy Service Programming Interfaces" on page 123
- "XACML Service" on page 123

## Authorization and Policy Service Overview

A *policy* is a rule that defines who is authorized to access a resource. A single policy can define authorization with either binary or non-binary decisions. (A binary decision is yes/no, true/false or allow/deny. A non-binary decision represents the value of an attribute; for example, a mail service might include a `mailboxQuota` attribute with a maximum storage value set for each user.) In general, the Policy Service allows administrators to configure, modify, and delete policies. The configured policies are then added to a *realm* and applied against the subjects in the realm. The Policy Service can be accessed using the Policy Service API: a privileged user can define access control policies using the administration API while a protected application or policy agent can obtain policy decisions using the evaluation API. The Policy Service relies on:

- A Policy Administration Point (PAP) implements the functionality to define policies. The Policy Service is the PAP.
- A Policy Enforcement Point (PEP) to protect an enterprise's resources by enforcing access control. The PEP uses the policy component of the Client SDK to retrieve policy decisions. The policy agent is the PEP.

- A Policy Decision Point (PDP) to evaluate policy and make an access determination. The Policy Service is the PDP.
- A data store in which configured policies are stored and from which they are retrieved. The Configuration Data Store is the data store.

Access to a resource is always preceded by a basic user session in which the requestor is authenticated, a session is created by the Authentication Service, and the session token identifier is validated. (See Chapter 6, "Models of the User Session and Single Sign-On Processes.") The policy agent protecting the resource then provides the session token identifier, resource name, desired action, and additional context parameters to the Policy Service which uses configured policies to determine if the user has been given permission to access the protected resource by a recognized authority. When the policy agent gets the decision from the Policy Service, it allows or denies access to the user, enforcing the policy decision provided by Policy Service. This whole process is referred to as *authorization*. The Policy Service is defined by the `amPolicy.xml` and, generally speaking:

- Provides a means for defining and managing access policies.
- Provides a means for defining custom policy plug-ins by providing names and class locations.
- Evaluates access policies.
- Acts as a PDP to deliver the result of a policy evaluation.
- Supports the delegation of policy management.
- Provides an SPI for extensibility.
- Provides access from remote clients using the Client SDK.
- Caches and reuses policy decisions, where applicable, to improve performance.
- Allows periodic polling of the Policy Service by a client to update locally cached policy decisions.
- Dynamically recognizes changes to policies and provides policy decisions that reflect them.

**Note –** The Policy Configuration Service provides a means to specify how policies are defined and evaluated. It enables you to specify, for example, which directory to use for subject lookup, the directory password, which search filters to use, and which subjects, conditions, and response providers to use. This configuration can be done within a realm or a subrealm and is accessible using the Federated Access Manager console.

See Chapter 5, "Managing Policies," in *Sun Federated Access Manager 8.0 Administration Guide* and Chapter 3, "Enforcing Authorization with the Policy Service," in *Sun Federated Access Manager 8.0 Developer's Guide* for more information.

# Policy Types

The Policy Service authorizes access to a user based on policies created and stored in the Federated Access Manager configuration data store. The following sections contain information on the two types of policies you can create.

- "Normal Policy" on page 119
- "Referral Policy" on page 122

For more information, see Chapter 5, "Managing Policies," in *Sun Federated Access Manager 8.0 Administration Guide*.

## Normal Policy

A *normal policy* specifies a protected resource and who is allowed to access it. The protected resource can be anything hosted on a protected server. Examples of protected resources are applications, document files, images, or the server itself. A normal policy consists of *rules*, *subjects*, *conditions*, and *response providers*. The following sections contain information on these elements.

- "Rules" on page 119
- "Subjects" on page 119
- "Conditions" on page 121
- "Response Providers" on page 122

### Rules

A *rule* defines the policy itself by specifying a resource, one or more sets of an action, and values for each action.

- A *resource* defines the specific object that is being protected. Examples of protected objects are an HTML page on a web site, or a user's salary information accessed using a human resources service.
- An *action* is the name of an operation that can be performed on the resource. Examples of web page actions are POST and GET. An allowable action for a human resources service might be `canChangeHomeTelephone`.
- A *value* defines the permission for the action. Examples are `allow` and`deny`.

### Subjects

A *subject* specifies the user or collection of users that the policy affects. The following list of subjects can be assigned to policies.

| | |
|---|---|
| Access Manager Identity Subjects | The identities you create and manage under the Subjects tab in a configured realm can be added as a value of the subject. |

| | |
|---|---|
| Authenticated Users | Any user with a valid session (even if they have authenticated to a realm that is different from the realm in which the policy is defined) is a member of this subject. This is useful if the resource owner would like to allow access to users from other organizations. To restrict a resource's access to members of a specific organization, use the Organization subject. |
| Web Services Clients | This implies that a web service client (WSC) identified by a session token identifier is a member of this subject — as long as the distinguished name (DN) of any principal contained in the session token identifier matches any selected value of this subject. |

The following list of subjects can only be specified after they are selected using the Policy Configuration Service of the appropriate realm.

| | |
|---|---|
| Federated Access Manager Roles | Any member of a Federated Access Manager role is a member of this subject. A Federated Access Manager role is created using Federated Access Manager running in legacy mode. These roles have object classes mandated by Federated Access Manager and can only be accessed through the hosting Federated Access Manager Policy Service.<br>**Note –** This subject can be used when connected to an AMSDK data store. |
| LDAP Groups | Any member of an LDAP group can be added as a value of this subject. |
| LDAP Roles | Any LDAP role can be added as a value of this subject. An LDAP Role is any role definition that uses the Sun Java System Directory Server role capability. These roles have object classes mandated by Directory Server role definition. The LDAP Role Search filter can be modified in the Policy Configuration Service to narrow the scope and improve performance. |
| LDAP Users | Any LDAP user can be added as a value of this subject. |
| Organization | Any member of a realm is a member of this subject. |

## Conditions

A *condition* specifies additional constraints that must be satisfied for a policy be applicable. For example, you can define a condition to limit a user's network access to a specific time period. The condition might state that the subject can access the network only between 7:00 in the morning and 10:00 at night. Federated Access Manager allows for the following list of conditions.

| | |
|---|---|
| Active Session Time | Sets a condition based on constraints configured for user session time such as maximum session time. |
| Authentication Chain | The policy is applicable if the user has successfully authenticated to the authentication chain in the specified realm. If the realm is not specified, authentication to any realm at the authentication chain will satisfy the condition. |
| Authentication Level | The Authentication Level attribute indicates the level of trust for authentication. The policy is applicable if the user's authentication level is greater than or equal to the Authentication Level set in the condition, or if the user's authentication level is less than or equal to the Authentication Level set in the condition, depending on the configuration. |
| Authentication Module Instance | The policy applies if the user has successfully authenticated to the authentication module in the specified realm. If the realm is not specified, authentication to any realm at the authentication module will satisfy the condition. |
| IP Address/DNS Names | Sets a condition based on a range of IP Addresses, or a DNS name. |
| Current Session Properties | Decides whether a policy is applicable to the request based on values set in the user's Access Manager session. |
| LDAP Filter Condition | The policy is applicable when the defined LDAP filter locates the user entry in the LDAP directory that was specified in the Policy Configuration service. |
| Realm Authentication | The policy applies if the user has authenticated to the specified realm. |
| Time | Sets the condition based on time constraints (time, day, date, time zone). |

### Response Providers

*Response providers* are plug-ins that provide policy response attributes. Policy response attributes typically provide values for attributes in the user profile. The attributes are sent with policy decisions to the PEP which, in turn, passes them in headers to an application. The application typically uses these attributes for customizing pages such as a portal page. Federated Access Manager includes one implementation of the `com.sun.identity.policy.interfaces.ResponseProvider` class, the `IDResponseProvider`. See Chapter 3, "Enforcing Authorization with the Policy Service," in *Sun Federated Access Manager 8.0 Developer's Guide* for more information.

## Referral Policy

A user with the Top—level Realm Administrator or Policy Administrator roles can create policy. (A Realm Administrator or Policy Administrator configured for a specific realm have permission to create policies only for resources delegated to that realm.) A *referral policy* enables either administrator to delegate policy configuration tasks. A referral policy delegates both policy creation and policy evaluation, and consists of one or more rules and one or more referrals.

- A *rule* defines the resource of which policy creation or evaluation is being referred.
- A *referral* defines the identity to which the policy creation or evaluation is being referred.

Referral policies delegate policy management privileges to another entity such as a peer realm, a subrealm, or even a third-party product. (You can implement custom referrals by using the Policy APIs.) For example, assume a top-level realm exists named `ISP`. It contains two subrealms: `company1` and `company2`. The Top-Level Realm Administrator for `ISP` can delegate policy management privileges so that a Realm Administrator in `company1` can create and manage policies only within the`company1` realm, and a Realm Administrator in `company2` can create and manage policies only within the `company2` realm. To do this, the Top-Level Realm Administrator creates two referral policies, defining the appropriate realm in the rule and the appropriate administrator in the referral. See "Creating Policies" in *Sun Federated Access Manager 8.0 Administration Guide* for more information.

# Realms and Access Control

When a user logs into an application, Federated Access Manager plug-ins retrieve all user information, authentication properties, and authorization policies that the Federated Access Manager framework needs to form a temporary, virtual user identity. The Authentication Service and the Policy Service use this virtual user identity to authenticate the user and enforce the authorization policies, respectively. All user information, authentication properties, and authorization policies is contained within a *realm*. You create a realm when you want to apply policies to a group of related subjects, services or servers. The Policy Configuration Service within the realm provides a means to specify how policies are defined and evaluated. It enables you to specify, for example, which directory to use for subject lookup, the directory password, which search filters to use, and which subjects, conditions, and response providers to use. For example, you can create a realm that groups all servers and services that are accessed regularly by employees in one region. And, within that regional grouping realm, you can group all servers and services accessed regularly by employees in a specific division such as Human Resources. A configured policy might state that all Human Resources administrators can access the URL `http://HR.example.com/HRadmins/index.html`. You might also add constraints to this policy: it is applicable only Monday through Friday from 9:00 a.m. through 5:00 p.m. Realms facilitate the delegation of policy management privileges. These configurations can be done within a realm or a sub realm and is accessible using the Federated Access Manager console.

---

**Note –** Access control realms can be configured to use any supported user database.

---

# Policy Service Programming Interfaces

Federated Access Manager provides both Java API and C API for writing clients that remote applications can use to administer policies and evaluate policy decisions. They are used to add, lookup, modify or replace policies, and to evaluate policy decisions when a principal attempts an action on a protected resource. Communication between the API and the Policy Service occurs by sending XML messages over HTTP(S). Additionally, you can extend and customize the Policy Service using the SPI. The classes are used by service developers and policy administrators who need to provide additional policy features as well as support for legacy policies. For example, you can develop customized plug-ins for creating custom policy subjects, referrals, conditions, and response providers. Lastly, the Client SDK is provided to implement policy evaluation logic on a remote web server or application server. For information, see Chapter 1, “Enhancing Remote Applications Using the Client Software Development Kit,” in *Sun Federated Access Manager 8.0 Developer’s Guide*, Chapter 3, “Enforcing Authorization with the Policy Service,” in *Sun Federated Access Manager 8.0 Developer’s Guide*, the *Sun Federated Access Manager 8.0 C API Reference*, and the *Federated Access Manager 8.0 Java API Reference*.

# XACML Service

eXtensible Access Control Markup Language (XACML) is a markup language that provides an XML syntax for defining policies (who can do what, where can it be done, and when), for querying whether access to a protected resource can be allowed (requests), and for receiving responses to those queries (decisions). XACML is built around the standard access control separation of the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP) as discussed in “Authorization and Policy Service Overview” on page 117 except you use XACML formatted queries and responses. The XACML PEP is responsible for intercepting all access requests, collecting the appropriate information (such as who is making the request, which resource is being accessed, and what action is to be taken), and sending a request for a decision to the XACML PDP. The XACML PDP (Federated Access Manager) evaluates configured policies against the information in the decision request. It uses a Context Handler to request the appropriate policies and attributes in order to render one of the following decisions.

- Permit
- Deny
- Not Applicable (no policy created by this PDP applies to the access request)
- Indeterminate (an error occurred that prevents the PDP from knowing the correct response)

The following sections contain more information.

- "XACML in Federated Access Manager" on page 124
- "XACML Programming Interfaces" on page 127

## XACML in Federated Access Manager

Federated Access Manager implements the SAML v2 Profile of XACML version 2.0 thus supporting `XACMLAuthzDecisionQuery` and `XACMLAuthzDecisionStatement`. In a Federated Access Manager XACML interaction, after receiving a request for access, the XACML PEP makes a `XACMLAuthzDecisionQuery` request and receives a `XACMLAuthzDecisionStatement` response that contains the decision. (The policies themselves are not returned.) The XACML components on the client side include Client SDK interfaces for passing XACML requests and receiving XACML responses as well as an interface to construct the communications.

---

**Note –** The framework relies internally on the Client SDK SAML v2 interfaces for communication between the PEP and PDP, and includes an implementation of the SAML v2 request handler called the `XACML2AuthzDecisionQueryHandler` that plugs into the SAML v2 Service framework.

---

The XACML components on the Federated Access Manager side include out-of-the-box implementations of XACML mappers for subjects, resources, actions and environment. These implementations use the Policy Service to compute authorization decisions. Figure 8–1 illustrates how XACML and Federated Access Manager interact with each other. The communications are explained more fully following the image.

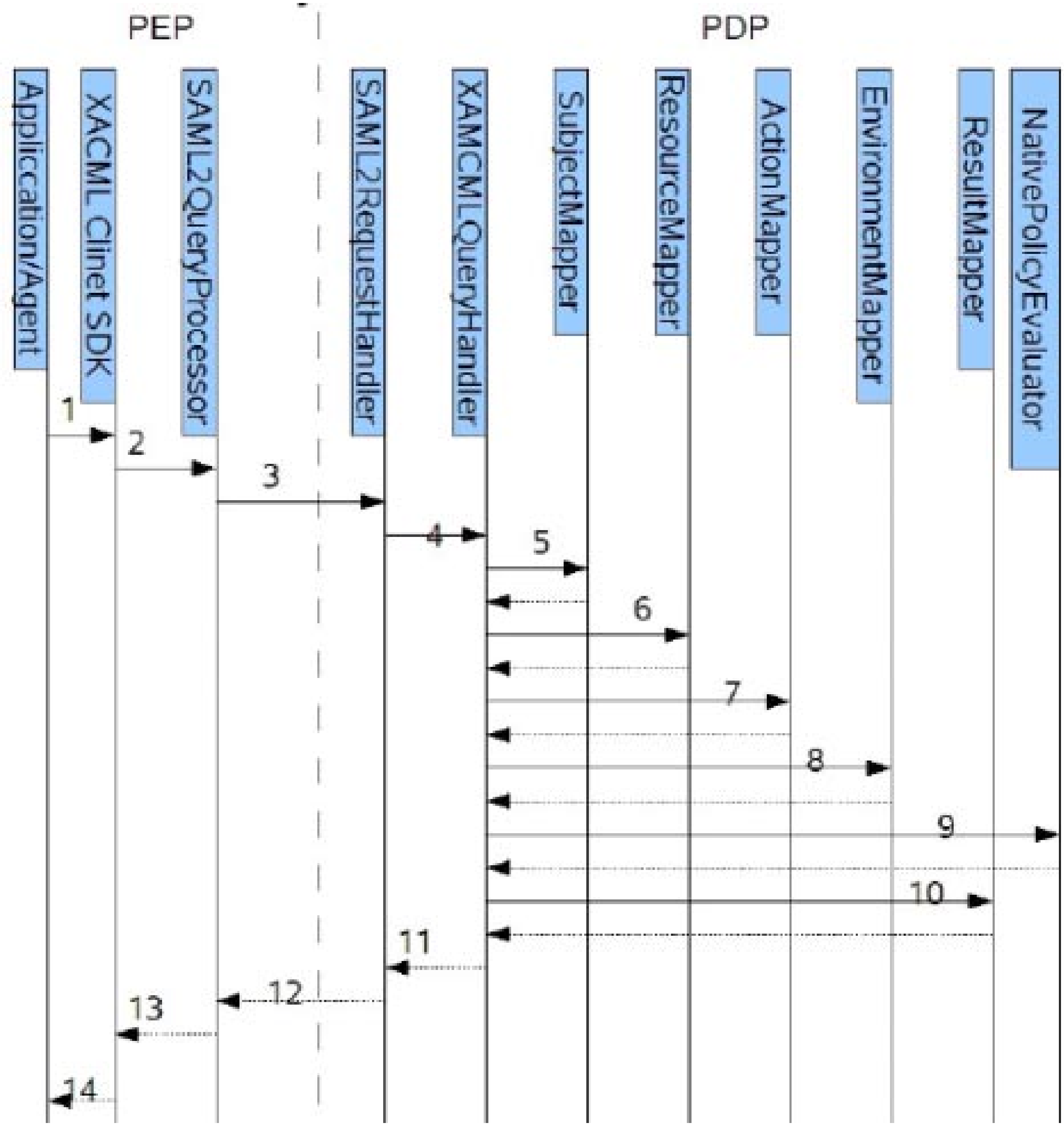

PEP
PDP
Appliccation/Agent
XACML Clinet SDK
SAML2QueryProcessor
SAML2RequestHandler
XAMCMLQueryHandler
SubjectMapper
ResourceMapper
ActionMapper
EnvironmentMapper
ResultMapper
NativePolicyEvaluator
1
2
3
4
5
6
7
8
9
10
11
12
13
14

FIGURE 8–1 XACML Process Flow

1. The policy agent protecting a resource constructs a XACML access request using the Client SDK.
2. The Client SDK wraps the request in a `XACMLAuthzDecisionQuery` element and sends it to the SAML v2 query processor on the local machine (also part of the Client SDK).
3. The SAML v2 query processor consults the metadata for the PEP and the PDP, sets additional elements or attributes in the query, signs it (if necessary) and sends a SOAP request containing the query to the PDP.
4. The SAML v2 request handler on the PDP side receives the request, consults the metadata for the PEP and the PDP, verifies the trust relationships, enforces any signing or encryption requirements, verifies the signature and forwards the query to the `XACMLAuthzDecisionQueryHandler`.
5. The `XACMLAuthzDecisionQueryHandler` consults the appropriate metadata using the `entityID` values of the PEP and PDP (included in the request) to find the correct mapper implementations to use.
6. `XACMLAuthzDecisionQueryHandler` uses the Resource mapper to map the given Resource to a resource and service configured with Federated Access Manager.
7. `XACMLAuthzDecisionQueryHandler` uses the Action mapper to map the given Action to an action name configured with Federated Access Manager.
8. `XACMLAuthzDecisionQueryHandler` uses the Environment mapper to map the given Environment to conditions configured with Federated Access Manager.
9. `XACMLAuthzDecisionQueryHandler` uses the Federated Access Manager policy evaluator to get the policy decision.
10. `XACMLAuthzDecisionQueryHandler` uses the Result mapper to map the decision to an XACML Result element.

---

**Note –** Federated Access Manager is not an XACML policy engine. It has no support for XACML policies themselves and thus no support for retrieving the policies, only the decision.

---

11. `XACMLAuthzDecisionQueryHandler` wraps the XACML Result in an XACML Response, the XACML Response in an `XACMLAuthzDecisionStatement`, the `XACMLAuthzDecisionStatement` in a SAML Assertion, the Assertion in a SAML Response, and hands over the SAML Response to the SAML v2 request handler.
12. The SAML v2 request handler sets additional attributes and elements (based on the SAML v2 protocol), signs it as required and returns it in a SOAP message to the PEP side.
13. The SAML v2 query processor verifies the trust relationships, the signing requirements, and the signature as necessary. It then extracts the SAML Response from the SOAP message and returns it to the XACML portion of the Client SDK.

14. The Client SDK extracts the XACML Response from the SAML v2 Response and returns it (and the decision) to the client application.

## XACML Programming Interfaces

Federated Access Manager provides Java API for using, and interacting with, the XACML Service. For information, see Chapter 1, “Enhancing Remote Applications Using the Client Software Development Kit,” in *Sun Federated Access Manager 8.0 Developer’s Guide*, Chapter 3, “Enforcing Authorization with the Policy Service,” in *Sun Federated Access Manager 8.0 Developer’s Guide*, and the *Federated Access Manager 8.0 Java API Reference*.

# PART III

# Federation Management Using Federated Access Manager

Sun Federated Access Manager provides a framework for implementing a federated identity infrastructure, enabling single sign-on, provisioning users dynamically, and sharing identity attributes across security domains. The chapters in this third part of the Sun Federated Access Manager Technical Overview contains information on federation management.

- Chapter 9, “What is Federation?”
- Chapter 10, “Federation Management with Federated Access Manager”
- Chapter 11, “Choosing a Federation Option”

◆ ◆ ◆

CHAPTER 9

# What is Federation?

Federation establishes a standards-based method for sharing and managing identity data and establishing single sign-on across security domains and organizations. It allows an organization to offer a variety of external services to trusted business partners as well as corporate services to internal departments and divisions. Forming trust relationships across security domains allows an organization to integrate applications offered by different departments or divisions within the enterprise as well as engage in relationships with cooperating business partners that offer complementary services. Towards this end, multiple industry standards, such as those developed by the Organization for the Advancement of Structured Information Standards (OASIS) and the Liberty Alliance Project, are supported. This chapter contains an overview of federation.

- "The Concept of Federation" on page 131
- "The Concept of Trust" on page 133
- "How Federation Works" on page 133

## The Concept of Federation

As a concept, federation encompasses both *identity federation* and *provider federation*.

- "Identity Federation" on page 131
- "Provider Federation" on page 132

### Identity Federation

In one dictionary, *identity* is defined as "a set of information by which one person is definitively distinguished." This information undoubtedly begins with the document that corroborates a person's name: a birth certificate. Over time, additional information further defines different aspects of an individual's identity. The composite of this data constitutes an identity with each specific piece providing a distinguishing characteristic. Each of the following represents data that designates a piece of a person's identity as it relates to the enterprise for which the data was defined.

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

Because the Internet is now one of the primary vehicles for the types of interactions represented by identity-defining information, people are creating online identities specific to the businesses with which they are interacting. By creating a user account with an identifier and password, an email address, personal preferences (such as style of music, or opt-in/opt-out marketing decisions) and other information specific to the particular business (a bank account number or ship-to address), a user is able to distinguish their account from others who also use the enterprise's services. This distinguishing information is referred to as a *local identity* because it is specific to the *service provider* (a networked entity that provides one or more services to other entities) for which it has been defined. Sending and receiving email, checking bank balances, finalizing travel arrangements, accessing utility accounts, and shopping are just a few online services for which a user might define a local identity. If a user accesses all of these services, many different local identities have been configured. Considering the number of service providers for which a user can define a local identity, accessing each one can be a time-consuming and frustrating experiencing. In addition, although most local identities are configured independently (and fragmented across the Internet), it might be useful to connect the information. For example, a user's local identity with a bank could be securely connected to the same user's local identity with a utility company for easy, online payments. This virtual phenomenon offers an opportunity for a system in which users can *federate* these local identities. *Identity federation* allows the user to link, connect, or bind the local identities that have been created for each service provider. The linked local identities, referred to as a *federated identity*, allow the user to log in to one service provider site and click through to an affiliated service provider without having to reauthenticate or reestablish identity; in effect, single sign-on (SSO).

## Provider Federation

Provider federation begins with a circle of trust. A *circle of trust* is a group of service providers who contractually agree to exchange authentication information. Each circle of trust must include at least one *identity provider*, a service provider that maintains and manages identity data, and provides authentication services. After the business contracts and policies defining a circle of trust are in place, the specific protocols, profiles, endpoints, and security mechanisms being used by each member is collected into a metadata document that is exchanged among all other members of the circle. Federated Access Manager provides the tools necessary to integrate the metadata and enable a circle of trust technologically. Authentication within this federation is honored by all membered providers.

---

**Note –** The establishment of contractual trust agreements between providers is beyond the scope of this guide. See "The Concept of Trust" on page 133 for an overview.

---

## The Concept of Trust

Federating identities assumes existing trust relationships between participants. This trust is usually defined through business arrangements or contracts that describe the technical, operational, and legal responsibilities of each party and the consequences for not completing them. When defined, a trust relationship allows one organization to trust the user authentication and authorization decisions of another organization. This trust then enables a user to log in to one site and, if desired, access a trusted site without reauthentication.

Ensure that trust agreements are in force before configuring circles of trust with Federated Access Manager and going live. The Liberty Alliance Project has created a support document for helping to establish these trust arrangements. The *Liberty Trust Model Guidelines* document is located on the Support Documents and Utility Schema Files page of the Liberty Alliance Project web site.

## How Federation Works

The goal of federation is to enable individuals and service providers to protect identity data while conducting network transactions across secure domains. When organizations form a trust agreement, they agree to exchange user authentication information using specific web technologies. The trust agreement would be among multiple service providers that offer web-based services to users and, at least, one *identity provider* (a service provider that maintains and manages identity information). Once *metadata* (a particular provider's federation configuration information) is exchanged and the trust is established technologically, single sign-on can be enabled between all the included providers, and users may opt to federate their multiple identities (depending on the protocol being used). In Federated Access Manager, the trust agreement is virtually configured as a *circle of trust* using the console or command line interface. A circle of trust contains *providers* (service providers or identity providers) that are grouped together for the purpose of offering identity federation. *Identity federation* occurs when a user chooses to unite distinct service provider and identity provider accounts while retaining the individual account information with each provider. The user establishes a link that allows the exchange of authentication information between provider accounts. Users can choose to federate any or all identities they might have. After identity federation, when a user successfully authenticates to one of the service providers, access to any of the federated accounts within the circle of trust is allowed *without having to reauthenticate*. The following figure shows the subjects involved in federation.

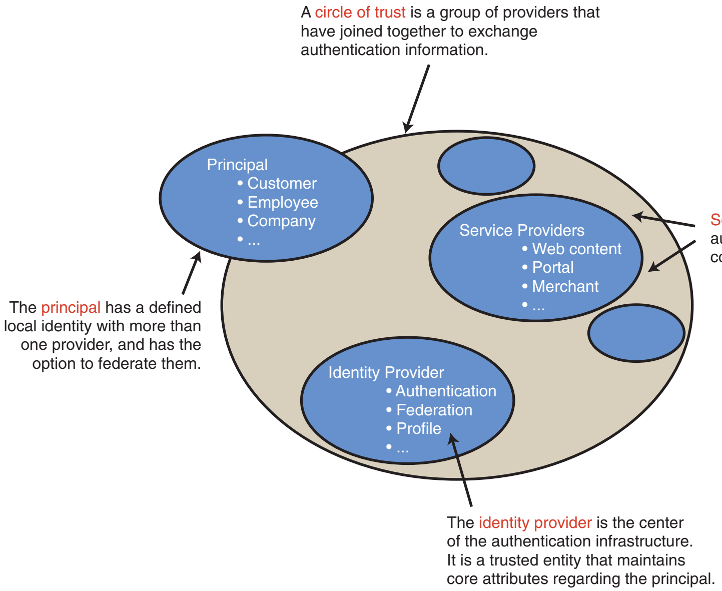

A circle of trust is a group of providers that have joined together to exchange authentication information.
Principal
• Customer
• Employee
• Company
• ...
Service Providers
• Web content
• Portal
• Merchant
• ...
Service providers in the authentication domain offer complimentary services.
The principal has a defined local identity with more than one provider, and has the option to federate them.
Identity Provider
• Authentication
• Federation
• Profile
• ...
The identity provider is the center of the authentication infrastructure. It is a trusted entity that maintains core attributes regarding the principal.

**FIGURE 9–1** Subjects Involved in an Identity Federation

- A *principal* can have a defined local identity with more than one provider, and it has the option to federate the local identities. The principal might be an individual user, a group of individuals, a corporation, or a component of the Liberty architecture.
- A *service provider* is a commercial or not-for-profit organization that offers a web-based service such as a news portal, a financial repository, or retail outlet.
- An *identity provider* is a service provider that stores identity profiles and offers incentives to other service providers for the prerogative of federating their user identities. Identity providers might also offer services above and beyond those related to identity profile storage.
- To support identity federation, all service providers and identity providers must join together into a *circle of trust*. A circle of trust must contain at least one identity provider and at least two service providers. (One organization may be both an identity provider and a service provider.) Providers in a circle of trust must first write trust agreements to define their relationships. A *trust agreement* is a contract between organizations that defines how the circle will work. For more information, see "The Concept of Trust" on page 133.

A travel portal is a good example of a circle of trust. Typically, a travel portal is a web site designed to help you access various travel-related services from one location. The travel portal forms a partnership with each service provider displayed on its web site. (This might include hotels, airlines, and car rental agencies.) The user registers with the travel portal which, in effect, is the identity provider for the circle of trust. After logging in, the user might click through to an airline service provider to look for a flight. After booking a flight, the user might click through to an accommodations service provider to look for a hotel. Because of the trust agreements previously established, the travel portal shares authentication information with the airline service provider, and the airline service provider with the accommodations service provider. The user moves from the hotel reservations web site to the airline reservations web site without having to reauthenticate. All of this is transparent to the user who must, depending on the underlying federation protocol, choose to federate any or all local identities. The following figure illustrates the travel portal example.

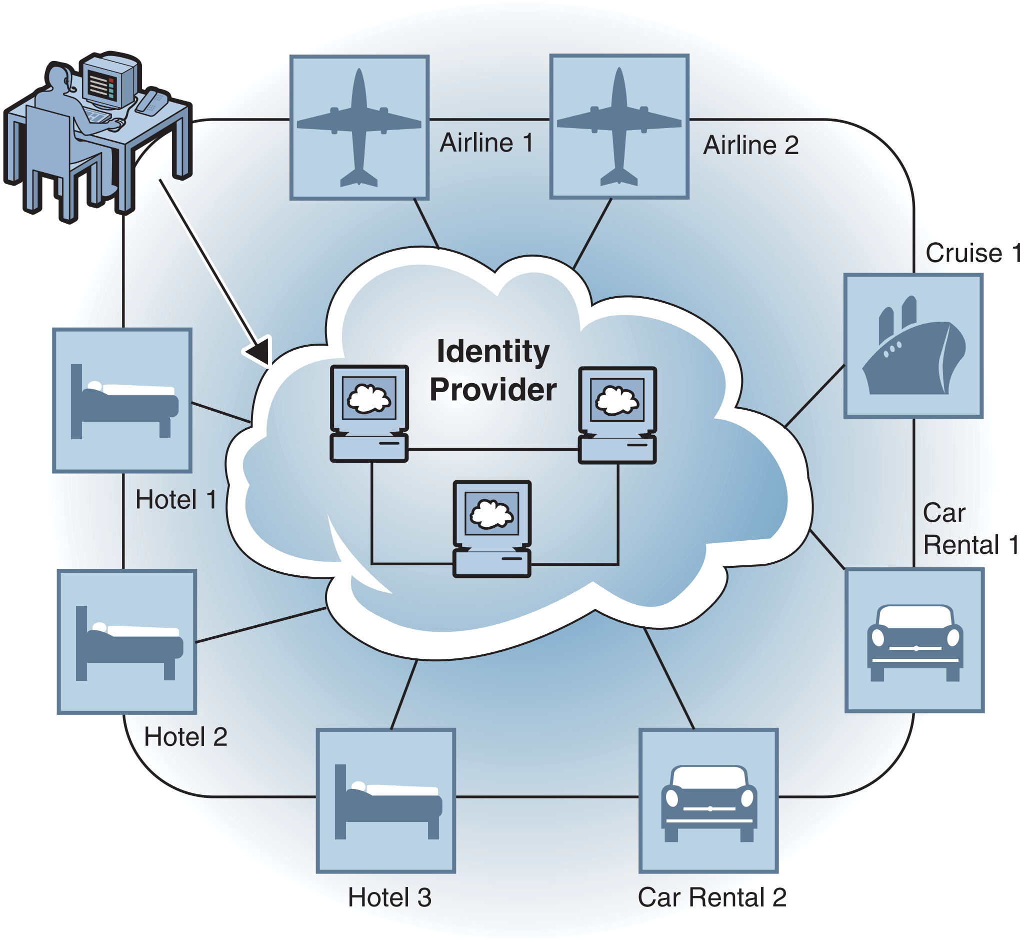

Airline 1
Airline 2
Cruise 1
Identity Provider
Hotel 1
Car Rental 1
Hotel 2
Hotel 3
Car Rental 2

**FIGURE 9–2** Federation Within a Travel Portal

# CHAPTER 10

# Federation Management with Federated Access Manager

Sun Federated Access Manager provides a pluggable framework for implementing federated identity infrastructures. The Federation framework places no restrictions on the use of network technologies, computer hardware, operating systems, programming languages or other hardware or software entities. It is based on, and conforms to, open industry standards to achieve interoperability among different vendors on heterogeneous systems, and provides the facility to log identity interactions and erroneous conditions. The following sections contain information about the federation framework.

- “Key Federation Management Features” on page 137
- “The Federation Framework Architecture” on page 142

## Key Federation Management Features

Federated Access Manager creates a comprehensive security and identity management framework optimized to work with, and extend, an identity provider's existing security infrastructure. The following list describes some key features:

- Exchange of credentials and security tokens across authentication domain partners for purposes of authentication and single sign-on.
- Automatic federation of user accounts across multiple security domains.
- Session management across authentication domains to determine when user interactions must be terminated (single logout).
- Import or export the data required to establish basic federated communication between providers.
- Manages and links providers that are available to participate in a circle of trust.
- Searches for available end points and identifies each provider's federation capabilities.
- Exchanges SAML security assertions among providers in a circle of trust.
- Data management choices include an LDAPv3 directory (OpenDS, Sun Java System Directory Server or Microsoft Active Directory).

- Included service provider interfaces (SPIs) to allow customized logic during the federation process.
- Support for bulk federation and auto federation.
- Support for The Fedlet, a web archive (WAR) of data that can be embedded into a service provider application.
- Support for Virtual Federation.
- Support for multiple federation protocols in one circle of trust.

The following sections contain additional information on the final three features listed.

- "The Fedlet" on page 138
- "Virtual Federation" on page 138
- "Multi-Federation Protocol Hub" on page 141

## The Fedlet

Fedlet is the name given to `fedlet.war`. The WAR is a very small archive of a few JARs, a keystore, and metadata (all stored in flat files) that can be embedded into a service provider's Java EE web application to allow for SSO between an identity provider instance of Federated Access Manager and the service provider application - WITHOUT installing Federated Access Manager on the service provider side. The service provider simply downloads the Fedlet, modifies their application to include the Fedlet JARs and, re-archives and redeploys the modified application. The service provider is now able to accept an HTTP POST (that contains a SAML v2 assertion) from the identity provider and retrieve included user attributes to accomplish SSO. (Currently, the Fedlet only supports the HTTP POST Profile.). XXXXXX LINK

## Virtual Federation

Virtual Federation provides a mechanism for one application to communicate identity information to a second application in a different domain. In essence, it provides a secure gateway that enables legacy applications to communicate authentication attributes without having to deal specifically with federation protocols and processing. Virtual Federation allows:

- Identity provider applications to push user authentication, profile and transaction information to a local instance of Federated Access Manager which then passes the data to a remote instance of Federated Access Manager at the service provider using federation protocols.
- Service provider applications to consume the received information.

Virtual Federation uses SAML v2 to transfer identity data between the communicating entities. The Client SDK (which contains Java and .NET interfaces) runs independent of Federated Access Manager and enables existing applications to handle the SAML v2 interactions. The following diagram illustrates this scenario.

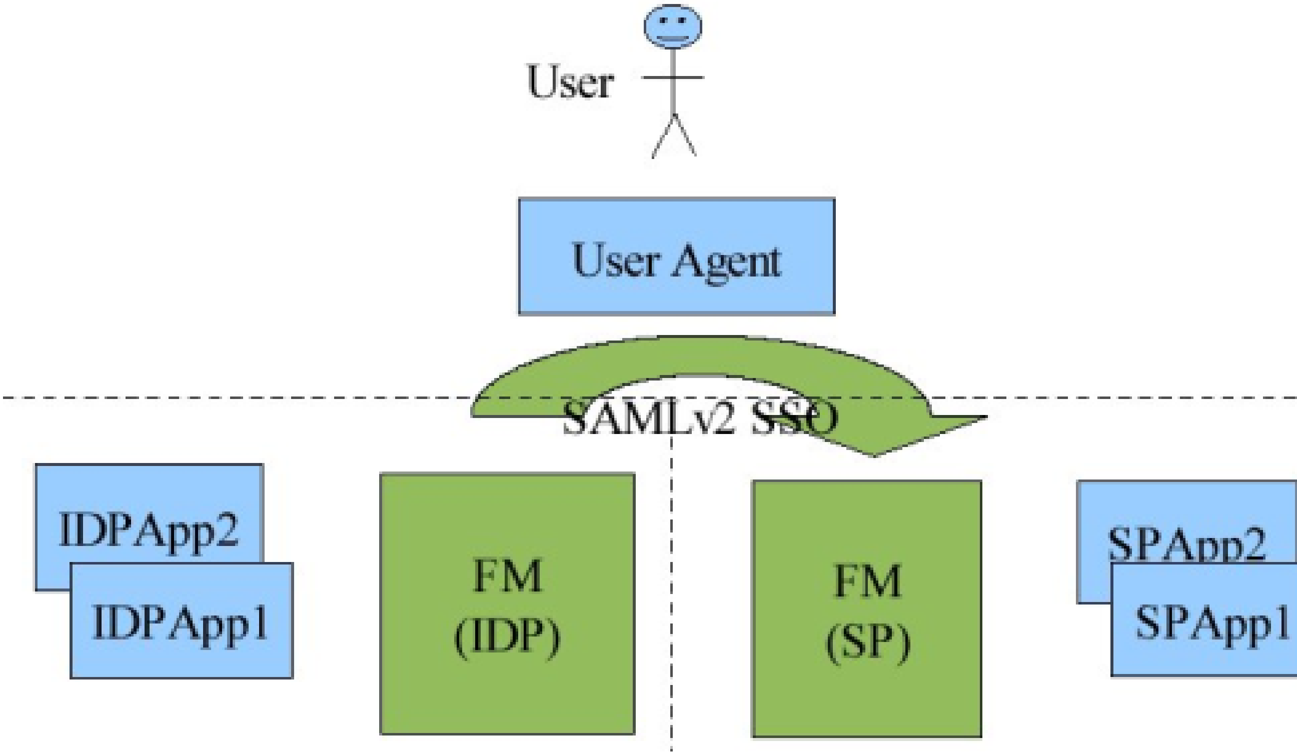


**FIGURE 10–1** Virtual Federation Architecture

The following use cases are applicable to Virtual Federation:

- "Authentication at Identity Provider" on page 139
- "Virtual Federation at Identity Provider" on page 140
- "Virtual Federation at Service Provider" on page 140
- "Global Single Logout" on page 140

## Authentication at Identity Provider

When a user is already authenticated within an enterprise, the legacy identity provider application sends a secure HTTP GET/POST message to Federated Access Manager asserting the identity of the user. Federated Access Manager verifies the authenticity of the message and establishes a session for the authenticated user. You can use Virtual Federation to transfer the user's authentication information to the local instance of Federated Access Manager in order to create the session.

### Virtual Federation at Identity Provider

When a user is already authenticated by, and attempts access to, a legacy identity provider application, the legacy application sends a secure HTTP POST message to the local instance of Federated Access Manager asserting the user's identity, and containing a set of attribute/value pairs related to the user (for example, data from the persistent store representing certain transactional states in the application). Federated Access Manager verifies the authenticity of the message, establishes a session for the authenticated user, and populates the session with the user attributes.

### Virtual Federation at Service Provider

When a user is already authenticated by the instance of Federated Access Manager at the identity provider and invokes an identity provider application that calls for redirection to a service provider, the identity provider invokes one of the previous use cases and encodes a SAML v2 SSO URL as a part of the request. The identity provider instance of Federated Access Manager then initiates SAML v2 SSO with the instance of Federated Access Manager at the service provider. The service provider's instance of Federated Access Manager then verifies the SAML v2 assertion and included attributes, and redirects to the service provider application, securely transferring the user attributes via a secure HTTP POST message. The service provider application consumes the attributes, establishes a session, and offers the service to the user.

### Global Single Logout

When a user is already authenticated and has established, for example, SSO with the instance of Federated Access Manager at the service provider, the user might click on a Global Logout link. The identity provider will then invalidate its local session (if created) and trigger SAML v2 single log out by invoking a provided Federated Access Manager URL. The Federated Access Manager identity provider executes the SAML v2 single log out, terminating the session on both provider instances of Federated Access Manager.

---

**Note –** An identity provider side application can initiate single logout by sending `sun.cmd=logout` attributes via a Virtual Federation interaction to a local instance of Federated Access Manager acting as the identity provider. In turn, this instance will execute SAML v2 single logout based on the current session.

---

For more information, see XXXXXX LINK.

## Multi-Federation Protocol Hub

Federated Access Manager allows a configured circle of trust to contain entities speaking different federation protocols thus supporting cross protocol single sign-on and logout among hosted identity providers in the same circle of trust. For example, you can create a circle of trust containing one identity provider instance that communicates with multiple federation protocol and three service provider instances that speak, respectively, Liberty ID-FF, SAML v2 and WS-Federation. Figure 10–2 illustrates the process of multi-federation protocol single sign-on and single logout.

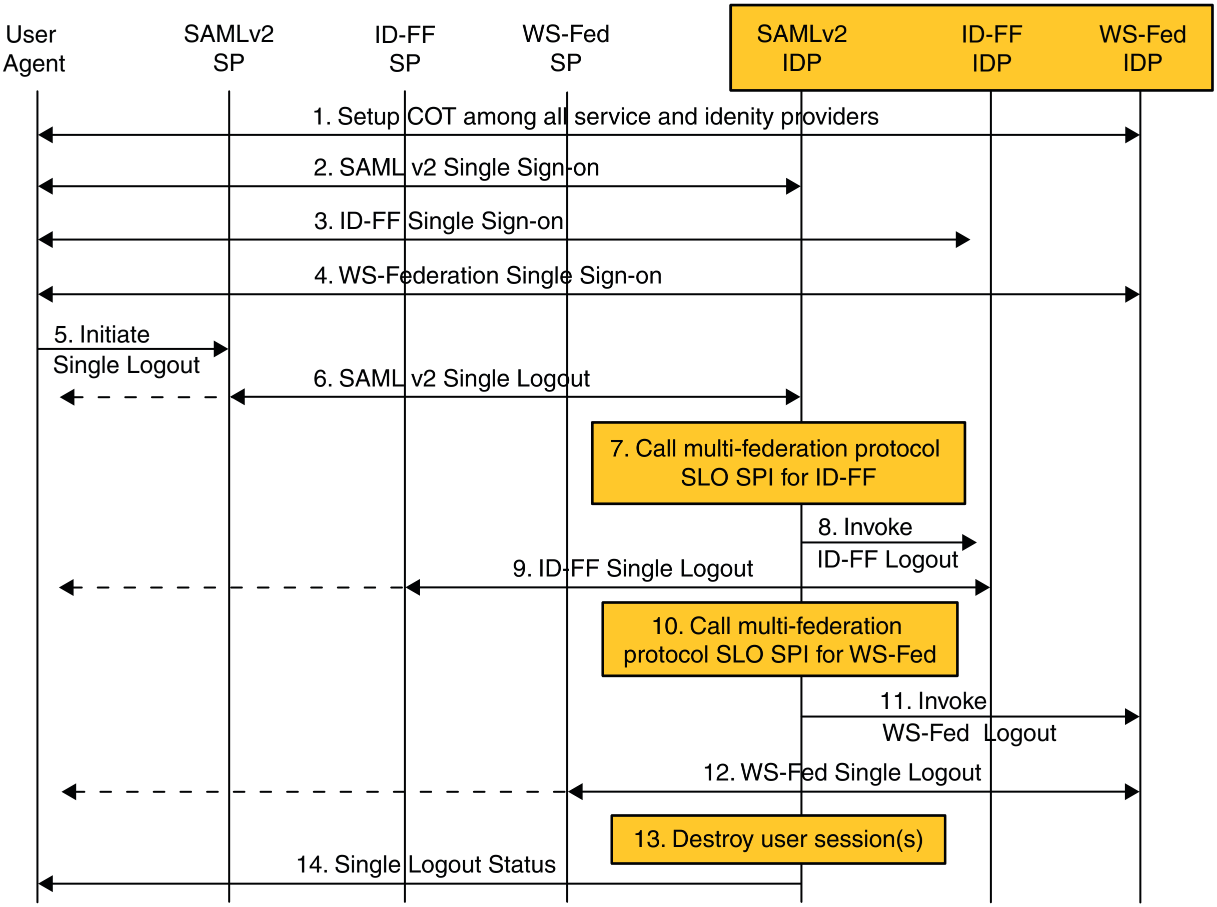

User Agent
SAMLv2 SP
ID-FF SP
WS-Fed SP
SAMLv2 IDP
ID-FF IDP
WS-Fed IDP
1. Setup COT among all service and idenity providers
2. SAML v2 Single Sign-on
3. ID-FF Single Sign-on
4. WS-Federation Single Sign-on
5. Initiate Single Logout
6. SAML v2 Single Logout
7. Call multi-federation protocol SLO SPI for ID-FF
8. Invoke ID-FF Logout
9. ID-FF Single Logout
10. Call multi-federation protocol SLO SPI for WS-Fed
11. Invoke WS-Fed Logout
12. WS-Fed Single Logout
13. Destroy user session(s)
14. Single Logout Status

**FIGURE 10–2** Multi-Federation Protocol Single Sign-on and Single Logout

For more information, see XXXXXX LINK.

# The Federation Framework Architecture

Federated Access Manager consists of web-based services [using SOAP, XML over HTTP(S) or HTML over HTTP(S)], and Java—based application provider interfaces (APIs) and service provider interfaces (SPIs). The figure below illustrates this architecture. Additionally, the figure shows an agent embedded into a web container. This agent enables the service provider applications to participate in the SAML or Liberty-based protocols. The darker boxes are components provided by Federated Access Manager.

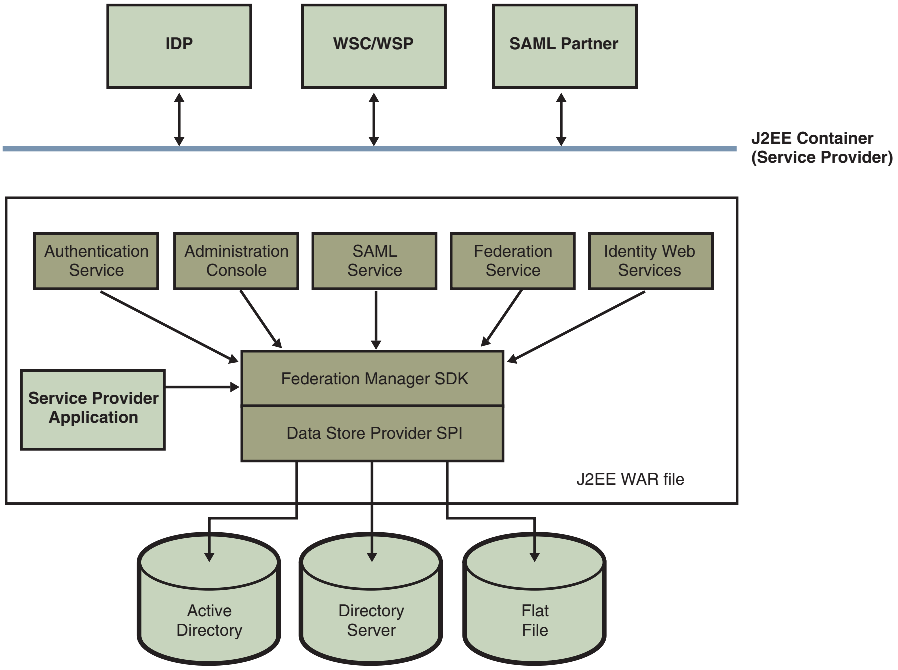

IDP
WSC/WSP
SAML Partner
J2EE Container
(Service Provider)
Authentication Service
Administration Console
SAML Service
Federation Service
Identity Web Services
Service Provider Application
Federation Manager SDK
Data Store Provider SPI
J2EE WAR file
Active Directory
Directory Server
Flat File

**FIGURE 10–3** Federation Framework Architecture

The components include:

| | |
|---|---|
| SAML Service | Provides SAML related services (versions 1.x and 2.0) including artifact and POST profile support, and assertion query support. |
| Federation Services | Provides services based on federation specifications. Features include federation and single sign-on, single logout, federation termination, name registration, and support for the Common Domain. Implemented web services include a SOAP binding service, a discovery service, a personal profile service, and an authentication service. |

| | |
|---|---|
| Authentication | Federated Access Manager provides a JAAS-based authentication framework. |
| Session | Federated Access Manager provides session management for service provider applications. |
| Logging | Federated Access Manager provides a logging service. It also provides activity logs for auditing. Audit logs can be stored in flat files or JDBC-compliant databases. |
| APIs | Includes a set of APIs for interaction between the SSO, logging, SAML, federation, and authentication components. Also included are APIs to build web services for clients and providers. |
| SPIs | Includes a set of Service Provider Interfaces (SPIs) into which applications can insert their custom logic. For instance, there is an SPI to do post federation processing, and an SPI for post processing after a successful single logout. |

◆ ◆ ◆

## CHAPTER 11

# Choosing a Federation Option

Sun Federated Access Manager supports multiple federation standards, such as those developed by the Organization for the Advancement of Structured Information Standards (OASIS) and the Liberty Alliance Project. This chapter contains information on these federation options.

- "Federation Options" on page 145
- "Using SAML" on page 146
- "Using the Liberty ID-FF" on page 154
- "Using WS-Federation" on page 164

## Federation Options

Federation is used to solve the problem of cooperation across heterogeneous, autonomous environments. In the beginning, federation meant using the Liberty Alliance Project Identity Federation Framework (Liberty ID-FF). Since then, other specifications have been developed with which federation can be accomplished including the Security Assertion Markup Language (SAML) and WS-Federation. To get started, use SAML v2 whenever possible as it supersedes both the Liberty ID-FF and SAML v1.x specifications. SAML v1.x was designed to address the issue of cross-domain single sign-on. It does not solve issues such as privacy, single logout, and federation termination. The Liberty Alliance Project was formed to develop technical specifications that would solve business process issues including single sign-on, account linking and consent, among others.

The SAML v1.x specifications and the Liberty Alliance Project specifications do not compete with one another. They are complementary. In fact, the Liberty Alliance Project specifications leverage profiles from the SAML specifications. The decision of whether to use SAML v1.x or the Liberty specifications depends on your goal. In general, SAML v1.x should suffice for single sign-on basics. The Liberty Alliance Project specifications can be used for more sophisticated functions and capabilities, such as global sign-out, attribute sharing, web services. The following table compares the benefits of the two.

**TABLE 11–1** Comparison of the SAML v1.x and Liberty Alliance Project Specifications

| SAML v1.x Uses | Liberty Alliance Project Uses |
|---|---|
| Cross-domain single sign-on | Single sign-on *only* after user federation |
| No user federation | User federation |
| No privacy control, best for use within one company | Built on top of SAML |
| User identifier is sent in plain text | User identifier is sent as a unique handle |

**Note –** Federated Access Manager has appropriated the terms from the Liberty ID-FF for all federation protocol implementations in the Federated Access Manager console.

The Liberty ID-FF and SAML v1.x should only be used when integrating with a partner that is not able to use SAML v2. If you are deploying Federated Access Manager with Microsoft Active Directory with Federation Services, you **must** use WS-Federation. More information on these options can be found in the following sections:

- "Using SAML" on page 146
- "Using the Liberty ID-FF" on page 154
- "Using WS-Federation" on page 164

# Using SAML

SAML defines an XML-based framework for exchanging identity information across security domains for purposes of authentication, authorization and single sign-on. It was designed to be used within other specifications (the Liberty Alliance Project, the Shibboleth project, and the Organization for the Advancement of Structured Information Standards have all adopted aspects of SAML) although the latest release (SAML v2) has incorporated back into the framework elements from the specifications developed by those very same organizations. The SAML specifications consist of a number of components, illustrated by Figure 11–1.

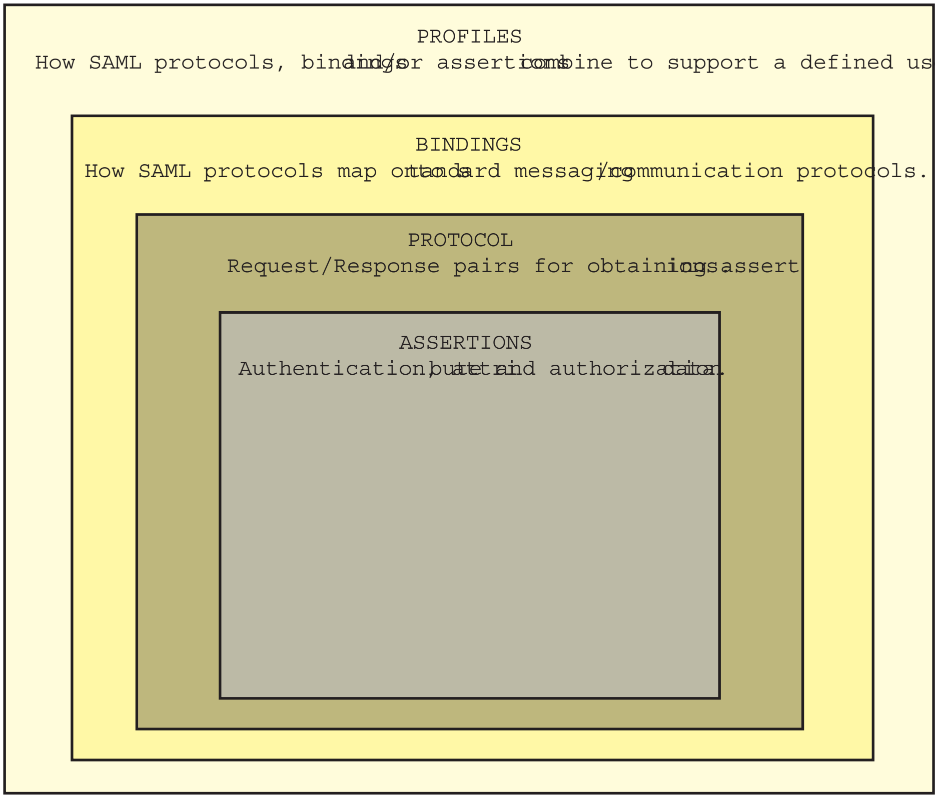

PROFILES
How SAML protocols, bindings and/or assertions combine to support a defined us
BINDINGS
How SAML protocols map onto standard messaging/communication protocols.
PROTOCOL
Request/Response pairs for obtaining assert
ASSERTIONS
Authentication, attribute and authorization data

**FIGURE 11–1** Components of the SAML Specifications

The SAML specification defines the *assertion* security token format as well as *profiles* that standardize the HTTP exchanges required to transfer XML requests and responses between an asserting authority and a trusted partner. An *assertion* is a package of verified security information that supplies one or more statements concerning a principal's authentication status, access authorization decisions, or identity attributes. (A person identified by an email address is a principal as might be a printer.) Assertions are issued by an asserting authority (a platform or application that declares whether a subject has been authenticated into its system), and received by relying parties (partner sites defined by the authority as *trusted*). Asserting authorities use different sources to configure assertion information, including external data stores or assertions that have already been received and verified.

The most recent SAML v2 specifications are defined more broadly than those developed for SAML v1.x — with particular attention paid to functionality dealing with federation. Before SAML v2 was introduced, SAML v1.x was simply a way to exchange identity data. In fact, up to version 1.1, the Liberty Alliance Project Identity Federation Framework (Liberty ID-FF) was developed using the SAML 1.0 specification. Liberty ID-FF version 1.2 was also developed using the SAML v1.1 specification. But, following the release of version 1.2, the Liberty ID-FF was incorporated into the SAML v2 specification. Additionally, SAML v2 adds components of the Shibboleth initiative. Thus, SAML v2 is a major revision, providing significant additional functionality and making the previous versions of SAML incompatible with it. Going forward, SAML v2 will be the basis on which Federated Access Manager implements federation. Figure 11–2 illustrates the convergence.

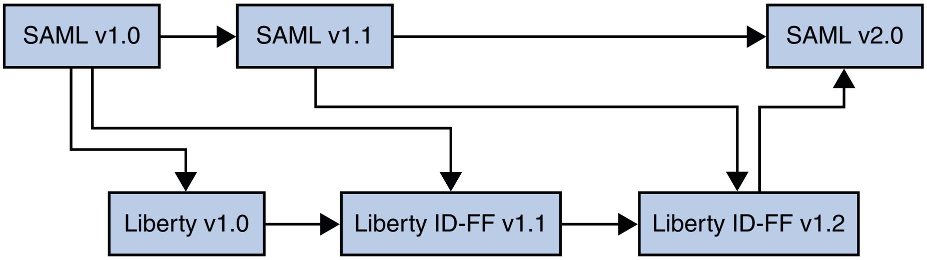


**FIGURE 11–2** Liberty ID-FF and SAML Convergence

---

**Note –** For more information on this convergence (including how the Shibboleth Project was also integrated), see the Federation section of Strategic Initiatives on the Liberty Alliance Project web site.

---

More information on the SAML implementations can be found in the following sections).

- "About SAML v2" on page 148
- "About SAML v1.x" on page 152
- "Using SAML or Federated Access Manager CDSSO" on page 153

---

**Caution –** SAML v1.x and SAML v2 assertions and protocol messages are incompatible.

---

## About SAML v2

Federated Access Manager delivers a solution that allows businesses to establish a framework for sharing trusted information across a distributed network of partners using the standards-based SAML v2. Towards this end, HTTP(S)-based service endpoints and SOAP service endpoints are supplied as well as assertion and protocol object manipulating classes. A web browser can access all HTTP(S)-based service endpoints and an application can make use of the SOAP endpoints and API as long as metadata for each participating business on BOTH sides of the SAML v2 interaction is exchanged beforehand.

Figure 11–3 illustrates the SAML v2 framework which consists of web-based services [using SOAP, XML over HTTP(S) or HTML over HTTP(S)], and Java™-based application provider interfaces (API) and service provider interfaces (SPI). Additionally, the figure shows an agent embedded into a web container in which a service provider application is deployed. This agent enables the service provider to participate in the SAML v1.x or Liberty ID-FF protocols.

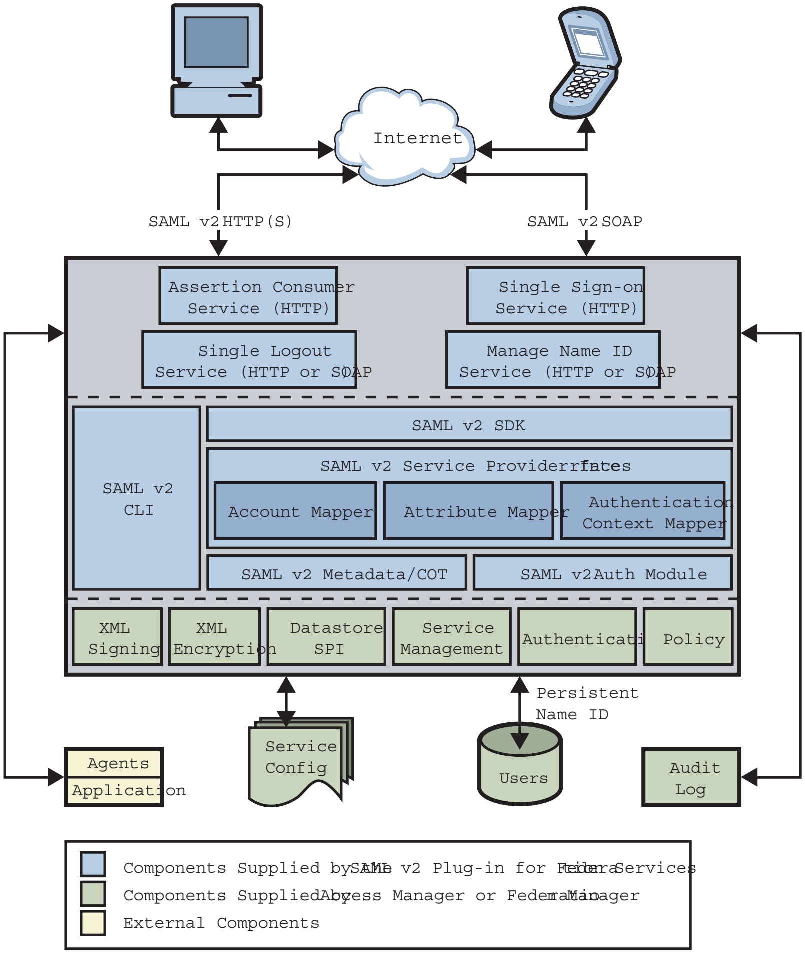

Internet
SAML v2 HTTP(S)
SAML v2 SOAP
Assertion Consumer Service (HTTP)
Single Sign-on Service (HTTP)
Single Logout Service (HTTP or SOAP)
Manage Name ID Service (HTTP or SOAP)
SAML v2 CLI
SAML v2 SDK
SAML v2 Service Provider Interfaces
Account Mapper
Attribute Mapper
Authentication Context Mapper
SAML v2 Metadata/COT
SAML v2 Auth Module
XML Signing
XML Encryption
Datastore SPI
Service Management
Authentication
Policy
Persistent Name ID
Agents
Application
Service Config
Users
Audit Log
Components Supplied by SAML v2 Plug-in for Federation Services
Components Supplied by Access Manager or Federation Manager
External Components

FIGURE 11–3 SAML v2 Architecture

The following sections contain more information about the SAML v2 framework.

- "Key Features" on page 150
- "Administration" on page 150
- "Application Programming Interfaces" on page 151

- “Service Provider Interfaces” on page 151
- “JavaServer Pages” on page 152

## Key Features

The key features of SAML v2 in Federated Access Manager include:

- Single sign-on using the POST profile, the Artifact binding (also referred to as HTTP redirect), and unsolicited responses (initiated by the identity provider)
- Single logout using HTTP redirect and SOAP binding
- Federation termination using HTTP redirect and SOAP binding
- Auto-federation (automatic linking of service provider and identity provider user accounts based on a common attribute)
- Bulk federation
- Dynamic creation of user accounts
- One time federation (transient NameID format for SSO)
- Basic Authentication, SSL and SSL with client authentication for SOAP binding security
- SAML v2 authentication
- Identity provider discovery
- XML verification, signing, encryption and decryption
- Profile initiation and processing using included JavaServer Pages™ (JSP™)
- Load balancing support
- Assertion failover
- XACML profile for authorization

  **Note –** See “XACML Service” on page 123 for more information.

- Protocol coexistence with the SAML v1.x and the Liberty ID-FF

## Administration

In order to communicate using the SAML v2 profiles you need, at least, two instances of Federated Access Manager. One instance will act for the identity provider and the other will act for the service provider. Name identifiers are used to communicate regarding a user.

---

**Note –** SAML v2 single sign-on interactions support both *persistent* and *transient* identifiers. A persistent identifier is saved to a particular user entry as the value of two attributes. A transient identifier is temporary and no data will be written to the user's data store entry.

---

To prepare your instances for SAML v2 interactions, you need to exchange a particular provider's configuration information or *metadata* between all participating identity and service providers, and assemble the providers into a *circle of trust*. Utility APIs can then be used to communicate with the data store, reading, writing, and managing the relevant properties and property values. For more information see the Chapter 10, “SAMLv2 Administration,” in *Sun Federated Access Manager 8.0 Administration Guide*.

## Application Programming Interfaces

The SAML v2 framework contains API that can be used to construct and process assertions, requests, and responses. The SAML v2 Java API packages include:

- The `com.sun.identity.saml2.assertion` package provides interfaces to construct and process SAML v2 assertions. It also contains the `AssertionFactory`, a factory class used to obtain instances of the objects defined in the assertion schema.
- The `com.sun.identity.saml2.common` package provides interfaces and classes used to define common SAML v2 utilities and constants.
- The `com.sun.identity.saml2.protocol` package provides interfaces used to construct and process the SAML v2 requests and responses. It also contains the `ProtocolFactory`, a factory class used to obtain object instances for concrete elements in the protocol schema.

More information can be found in “Using the SAML v2 SDK” in *Sun Federated Access Manager 8.0 Developer’s Guide* and the *Federated Access Manager 8.0 Java API Reference*.

## Service Provider Interfaces

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

- Account mapping (map between the account referred to in the incoming request and the local user account)
- Attribute mapping (specifies which set of user attributes in an identity provider user account needs to be included in an assertion, and maps the included attributes to attributes in the user account defined by the service provider)
- Authentication context mapping (map between Authentication Contexts defined in the SAML v2 specifications and authentication framework schemes defined in Federated Access Manager (user/module/service/role/level based authentication)

More information can be found in “Service Provider Interfaces” in *Sun Federated Access Manager 8.0 Developer’s Guide* and the *Federated Access Manager 8.0 Java API Reference*.

### JavaServer Pages

The SAML v2 framework provides JSP that can be used to initiate single sign-on, single logout and termination requests from either the identity provider or the service provider using a web browser. The JSP accept query parameters to allow flexibility in constructing SAML v2 requests; they can be modified for your deployment. More information can be found in “JavaServer Pages” in *Sun Federated Access Manager 8.0 Developer’s Guide*.

## About SAML v1.x

Federated Access Manager can be configured to use SAML v1.x to achieve interoperability between vendor platforms that provide SAML v1.x assertions. Assertions are issued by a SAML v1.x asserting authority (a platform or application that declares whether a subject has been authenticated into its system), and received by relying parties (partner sites defined by the authority as *trusted*). SAML v1.x authorities use different sources to configure the assertion information, including external data stores or assertions that have already been received and verified. SAML v1.x can be used to allow Federated Access Manager to:

- Authenticate users and access trusted partner sites without having to reauthenticate; in effect, single sign-on.
- Act as a policy decision point (PDP), allowing external applications to access user authorization information for the purpose of granting or denying access to their resources. For example, employees of an organization can be allowed to order office supplies from suppliers if they are authorized to do so.
- Act as both an attribute authority that allows trusted partner sites to query a subject’s attributes, and an authentication authority that allows trusted partner sites to query a subject’s authentication information.
- Validate parties in different security domains for the purpose of performing business transactions.
- Build Authentication, Authorization Decision, and Attribute Assertions using the SAML v1.x API.
- Permit an XML-based digital signature signing and verifying functionality to be plugged in.

---

**Note –** Although Liberty ID-FF (as described in “Using the Liberty ID-FF” on page 154) integrates aspects of the SAML v1.x specifications, its usage of SAML v1.x is independent of the SAML v1.x framework as described in this section.

---

Figure 11–4 illustrates how SAML v1.x interacts with the other components in Federated Access Manager.

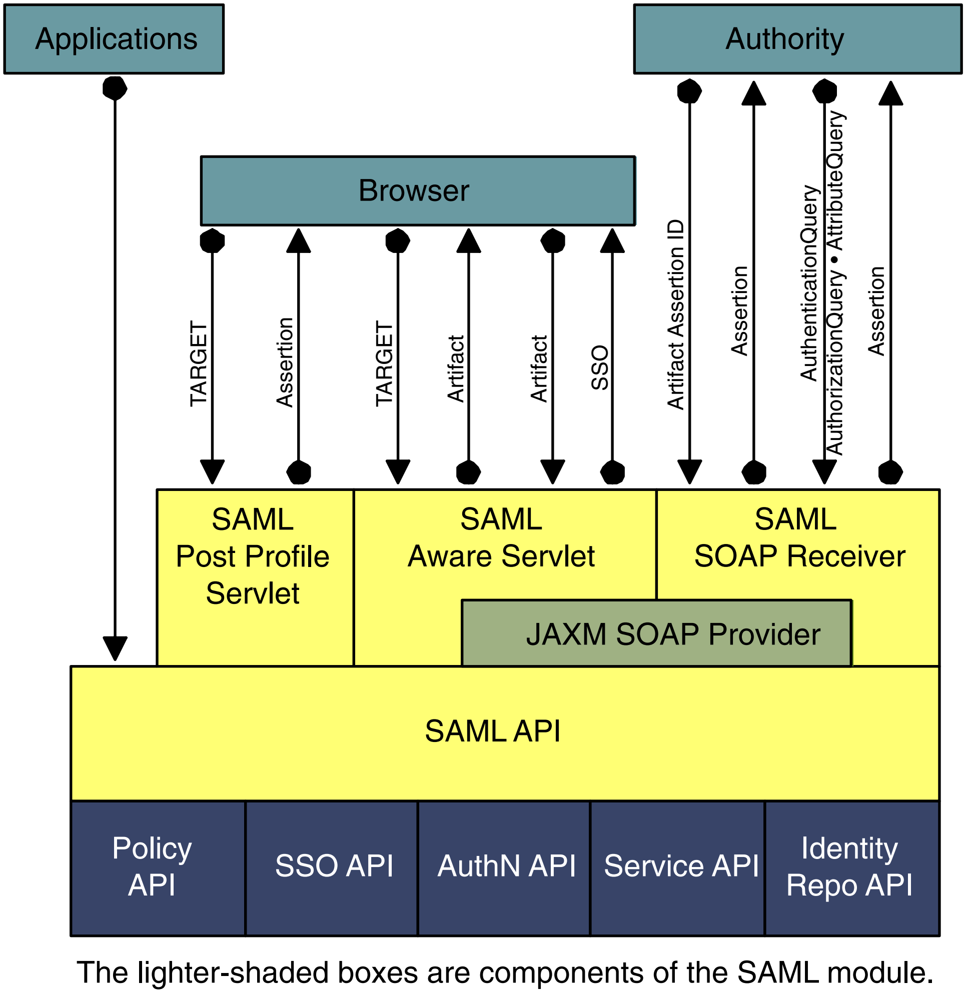

Applications
Authority
Browser
TARGET
Assertion
TARGET
Artifact
Artifact
SSO
Artifact Assertion ID
Assertion
AuthenticationQuery
AuthorizationQuery • AttributeQuery
Assertion
SAML Post Profile Servlet
SAML Aware Servlet
SAML SOAP Receiver
JAXM SOAP Provider
SAML API
Policy API
SSO API
AuthN API
Service API
Identity Repo API
The lighter-shaded boxes are components of the SAML module.

**FIGURE 11–4** SAML v1.x Interaction in Federated Access Manager

## Using SAML or Federated Access Manager CDSSO

Cross Domain Single Sign On (CDSSO) is a proprietary mechanism from Sun Federated Access Manager, designed before SAML and the Liberty Alliance Project specifications existed. CDSSO is still available and, in certain cases, is easier to set up and manage than the latter two specifications — which solve a broader set of SSO issues than CDSSO. CDSSO requires all policy agents to be configured to use a single Federated Access Manager server. This means only one user identity can exist in the entire system whereas, when using SAML/Liberty Alliance

Project, user identities can exist on multiple systems (service providers or identity providers). Because of the single identity in CDSSO interactions, issues such as account mapping, attribute flow and session synchronization are not relevant thus, if you need to implement these features, use SAML. If the following points are valid to your planned deployment, CDSSO may be a simpler and more suitable solution than SAML/Liberty Alliance Project:

- Only Sun Federated Access Manager and Sun policy agents are involved.
- Sun policy agents are configured to use the same Federated Access Manager infrastructure where multiple instances can exist.
- Federated Access Manager uses a single user identity store.
- Multiple instances of Federated Access Manager (configured for high-availability) must reside in a single DNS domain. Only policy agents can reside in different DNS domains.

For more information on CDSSO, see Chapter 6, "Models of the User Session and Single Sign-On Processes."

# Using the Liberty ID-FF

The Liberty Alliance Project was formed to develop technical specifications that would solve business process issues including single sign-on, federation and consent. The Liberty Alliance Project Identity Federation Framework (Liberty ID-FF) uses a name identifier to pass identity data between identity providers and service providers. The *name identifier* is a randomly generated character string that is assigned to a principal and used to federate the principal's accounts at the identity provider and service provider sites. This pseudonym allows all providers to identify a principal without knowing the user's actual identity. The name identifier has meaning only in the context of the relationship between providers. SAML v1.x is used for provider interaction.

---

**Note –** Liberty ID-FF was initially defined as an extension of SAML 1.0 (and later SAML 1.1). The extensions have now been contributed back into SAML v2 which, going forward, will be the basis on which the Liberty Alliance Project builds additional federated identity applications. See "Using SAML" on page 146 for more information on this convergence.

---

The following sections contain information about the Liberty ID-FF and the features implemented in Federated Access Manager.

- "Liberty ID-FF Features" on page 155
- "About the Liberty ID-FF Process" on page 161

# Liberty ID-FF Features

The following sections contain information about the Liberty ID-FF features implemented in Federated Access Manager.

- "Federated Single Sign-On" on page 155
- "Authentication and Authentication Context" on page 157
- "The Common Domain for Identity Provider Discovery" on page 159
- "Identifiers and Name Registration" on page 160
- "Global Logout" on page 161
- "Dynamic Identity Provider Proxying" on page 161

## Federated Single Sign-On

Let's assume that a principal has separate user accounts with a service provider and an identity provider in the same circle of trust. In order to gain access to these individual accounts, the principal would authenticate with each provider separately. If federating with the Liberty ID-FF though, after authenticating with the service provider, the principal may be given the option to federate the service provider account with the identity provider account. Consenting to the federation of these accounts links them for SSO, the means of passing a user's credentials between applications without the user having to reauthenticate. SSO and federated SSO have different processes. With Federated Access Manager, you can achieve SSO in the following ways:

- Install a policy agent in a web container to protect the application and pass the `HTTP_HEADER` and `REMOTE_USER` variables to the application to capture the user credentials. You may or may not need a custom authentication module.
- Customize the application's authentication module to create an `SSOToken` from the request object or from the SSO cookie. Afterwards, retrieve the user credentials using the SSO API and create a session data structure using the application's API.

To set up federated SSO, you must first establish SSO. Following that, enable federation in the metadata for the service provider entity and the identity provider entity using Federated Access Manager. Liberty ID-FF providers differentiate between federated users by defining a unique *identifier* for each account. (They are not required to use the principal's actual provider account identifier.) Providers can also choose to create multiple identifiers for a particular principal. However, identity providers must create one handle per user for service providers that have multiple web sites so that the handle can be resolved across all of them.

---

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

---

Federated Access Manager can accommodate the following SSO and federation-related functions:

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

Additionally, Federated Access Manager can accommodate the federation features explained in the following sections.

- "Auto-Federation" on page 156
- "Bulk Federation" on page 156

## Auto-Federation

Auto federation will automatically federate a user's disparate provider accounts based on a common attribute. During SSO, if it is deemed a user at provider A and a user at provider B have the same value for the defined common attribute (for example, an email address), the two accounts will be federated without consent or interaction from the principal. For more information, see "Auto-Federation" in *Sun Federated Access Manager 8.0 Administration Guide*.

## Bulk Federation

Federating one user's service provider account with their identity provider account generally requires the principal to visit both providers and link them. An organization though needs the ability to federate user accounts behind the scenes. Federated Access Manager provides a script for federating user accounts in bulk. The script allows the administrator to federate many (or all) of a principal's provider accounts based on metadata passed to the script. Bulk federation is useful when adding a new service provider to an enterprise so you can federate a group of existing employees to the new service. For more information, see "Bulk Federation" in *Sun Federated Access Manager 8.0 Administration Guide*.

## Authentication and Authentication Context

SSO is the means by which a provider of either type can convey to another provider that a principal has been authenticated. Authentication is the process of validating user credentials; for example, a user identifier accompanied by an associated password. You can authenticate users with Federated Access Manager in the following ways:

- Use a policy agent to insert HTTP header variables into the request object. This functions for web applications only.
- Use the authentication API to validate and retrieve user identity data. This will work with either web or non-web applications.

Identity providers use local (to the identity provider) session information mapped to a user agent as the basis for issuing SAML authentication assertions to service providers. Thus, when the principal uses a user agent to interact with a service provider, the service provider requests authentication information from the identity provider based on the user agent's session information. If this information indicates that the user agent's session is presently active, the identity provider will return a positive authentication response to the service provider. Federated Access Manager allows providers to exchange the following minimum set of authentication information with regard to a principal.

- Authentication status (active or not)
- Instant (time authenticated)
- Authentication method
- Pseudonym (temporary or persistent)

SAML v1.x is used for provider interaction during authentication but not all SAML assertions are equal. Different authorities issue SAML assertions of different quality. Therefore, the Liberty ID-FF defines how the consumer of a SAML assertion can determine the amount of assurance to place in the assertion. This is referred to as the *authentication context*, information added to the SAML assertion that gives the assertion consumer the details they need to make an informed entitlement decision. For example, a principal uses a simple identifier and a self-chosen password to authenticate to a service provider. The identity provider sends an assertion to a second service provider that states how the principal was authenticated to the first service provider. By including the authentication context, the second service provider can place an appropriate level of assurance on the associated assertion. If the service provider were a bank, they might require stronger authentication than that which has been used and respond to the identity provider with a request to authenticate the user again using a more stringent context. The authentication context information sent in the assertion might include:

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

- The authentication mechanisms (for example, password or smart card with PIN).

The Liberty ID-FF specifications define authentication context *classes* against which an identity provider can claim conformance. The Liberty ID-FF authentication contexts are listed and described in the following table.

**TABLE 11–2** Authentication Context Classes

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

**TABLE 11–2** Authentication Context Classes *(Continued)*

| Class | Description |
|---|---|
| Time-Sync-Token | Identified when a principal authenticates through a time synchronization token. |

For more information, see the *Liberty ID-FF Authentication Context Specification* and . Additionally, there is an XML schema defined which the identity provider authority can use to incorporate the context of the authentication in the SAML assertions it issues.

## The Common Domain for Identity Provider Discovery

Service providers need a way to determine which identity provider in a circle of trust is used by a principal requesting authentication. Because circles of trust are configured without regard to their location, this function must work across DNS-defined domains. A common domain is configured, and a common domain cookie written, for this purpose.

Let's suppose a circle of trust contains more than one identity provider. In this case, a service provider trusts more than one identity provider so, when a principal needs authentication, the service provider with which the principal is communicating must have the means to determine the correct identity provider. To ascertain a principal's identity provider, the service provider invokes a protocol exchange to retrieve the *common domain cookie*, a cookie written for the purpose of *introducing* the identity provider to the service provider. If no common domain cookie is found, the service provider will present a list of trusted identity providers from which the principal can choose. After successful authentication, the identity provider writes (using the configured Writer Service URL) a common domain cookie and, the next time the principal attempts to access a service, the service provider finds and reads the common domain cookie (using the configured Reader Service URL), to determine the identity provider. More information on the Common Domain for Identity Provider Discovery is available in the following sections, and in "Common Domain Services for Federation Management" in *Sun Federated Access Manager 8.0 Administration Guide*.

- "The Common Domain" on page 159
- "The Common Domain Cookie" on page 160
- "The Writer Service and the Reader Service" on page 160

### The Common Domain

The *common domain* is established for use only within the scope of identity provider discovery in a defined circle of trust. In Federated Access Manager deployments, the identity provider discovery WAR is deployed in a web container installed in a predetermined and preconfigured *common* domain so that the common domain cookie is accessible to all providers in the circle of trust. For example, if an identity provider is available at `http://www.Bank.com`, a service provider is available at `http://www.Store.com`, and the defined common domain is `RetailGroup.com`, the addresses will be `Bank.RetailGroup.com` and `Store.RetailGroup.com`, respectively. If the HTTP server in the common domain is operated by the service provider, the service provider will redirect the user agent to the appropriate identity provider.

### The Common Domain Cookie

After an identity provider authenticates a principal, the identity provider sets a URL-encoded cookie defined in a predetermined domain common to all identity providers and service providers in the circle of trust. The *common domain cookie* is named `_liberty_idp` for Liberty ID-FF and `_saml_idp` for SAML v2. After successful authentication, a principal's identity provider appends their particular encoded identifier to a list in the cookie. If their identifier is already present in the list, the identity provider may remove the initial appearance and append it again. The intent is that the service provider reads the last identifier on the cookie's list to find the principal's most recently established identity provider.

---

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

---

### The Writer Service and the Reader Service

After a principal authenticates with a particular identity provider, the identity provider redirects the principal's browser to the configured Writer Service URL using a parameter that indicates they are the identity provider for this principal. The Writer Service then writes a cookie using the parameter. Thereafter, all providers configured in this common domain will be able to tell which identity provider is used by this principal. Thus, the next time the principal attempts to access a service hosted by a service provider in the same common domain, the service provider retrieves and reads the common domain cookie, using the configured Reader Service URL, to determine the identity provider.

The Writer Service URL and the Reader Service URL can be defined for use with the Liberty ID-FF or the SAML v2 federation protocol. The URLs are defined when you create a circle of trust for federation. The Common Domain for Identity Provider Discovery for Liberty ID-FF is based on the Identity Provider Introduction Profile detailed in the *Liberty ID-FF Bindings and Profiles Specifications*. The Common Domain for Identity Provider Discovery for SAML v2 is an implementation of the Identity Provider Discovery Profile as described in the Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0 specification.

## Identifiers and Name Registration

Federated Access Manager supports name identifiers that are unique across all providers in a circle of trust. This identifier can be used to obtain information for or about the principal without requiring the user to consent to a long-term relationship with the service provider. When beginning federation, the identity provider generates an opaque value that serves as the initial name identifier that both the service provider and the identity provider use to refer to the principal when communicating with each other. After federation, the identity provider or the service provider may register a different opaque value. If a service provider registers a different opaque value for the principal, the identity provider must use the new identifier when communicating with the service provider about the principal. The reasons for changing an identifier would be implementation-specific. The initial name identifier defined by the identity provider is always used to refer to the principal unless a new name identifier is registered.

### Global Logout

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

### Dynamic Identity Provider Proxying

An identity provider can choose to proxy an authentication request to an identity provider in another authentication domain if it knows that the principal has been authenticated with this identity provider. The proxy behavior is defined by the local policy of the proxying identity provider. However, a service provider can override this behavior and choose not to proxy. This function can be implemented as a form of authentication when, for instance, a roaming mobile user accesses a service provider that is not part of the mobile home network. For more information see “Dynamic Identity Provider Proxying” in *Sun Federated Access Manager 8.0 Administration Guide*.

## About the Liberty ID-FF Process

The Liberty ID-FF is designed to work with heterogeneous platforms, various networking devices (including personal computers, mobile phones, and personal digital assistants), and emerging technologies. The process of Liberty ID-FF federation begins with authentication. A user attempting to access a resource protected by Federated Access Manager are redirected to the proprietary Authentication Service via an Federated Access Manager login page. After the user provides credentials, the Authentication Service allows or denies access to the resource based on the outcome.

---

**Note –** For more information about the proprietary Authentication Service, see the Chapter 7, “Authentication and the Authentication Service.”

---

When the user attempts access to a resource that belongs to a trusted member provider of a configured circle of trust, the process of user authentication begins with the search for a valid Federated Access Manager session token from the proprietary Authentication Service. The process can go in one of two directions based on whether a session token is found.

- If no session token is found, the principal is redirected to a location defined by the pre-login URL to establish a valid session.
- If a session token is found, the principal is granted (or denied) access to the requested page. Assuming access is granted, the requested page contains a link so the principal may federate the Federated Access Manager identity with the identity local to the requested site. If the principal clicks this link, federation begins.

Figure 11–5 illustrates these divergent paths. The process shown is the default process when no application has been deployed. When an application is deployed and using Federated Access Manager, the process will change based on the query parameters and preferences passed to Federated Access Manager from the participating application. For more information, see "The Pre-login URL" in *Sun Federated Access Manager 8.0 Administration Guide*.

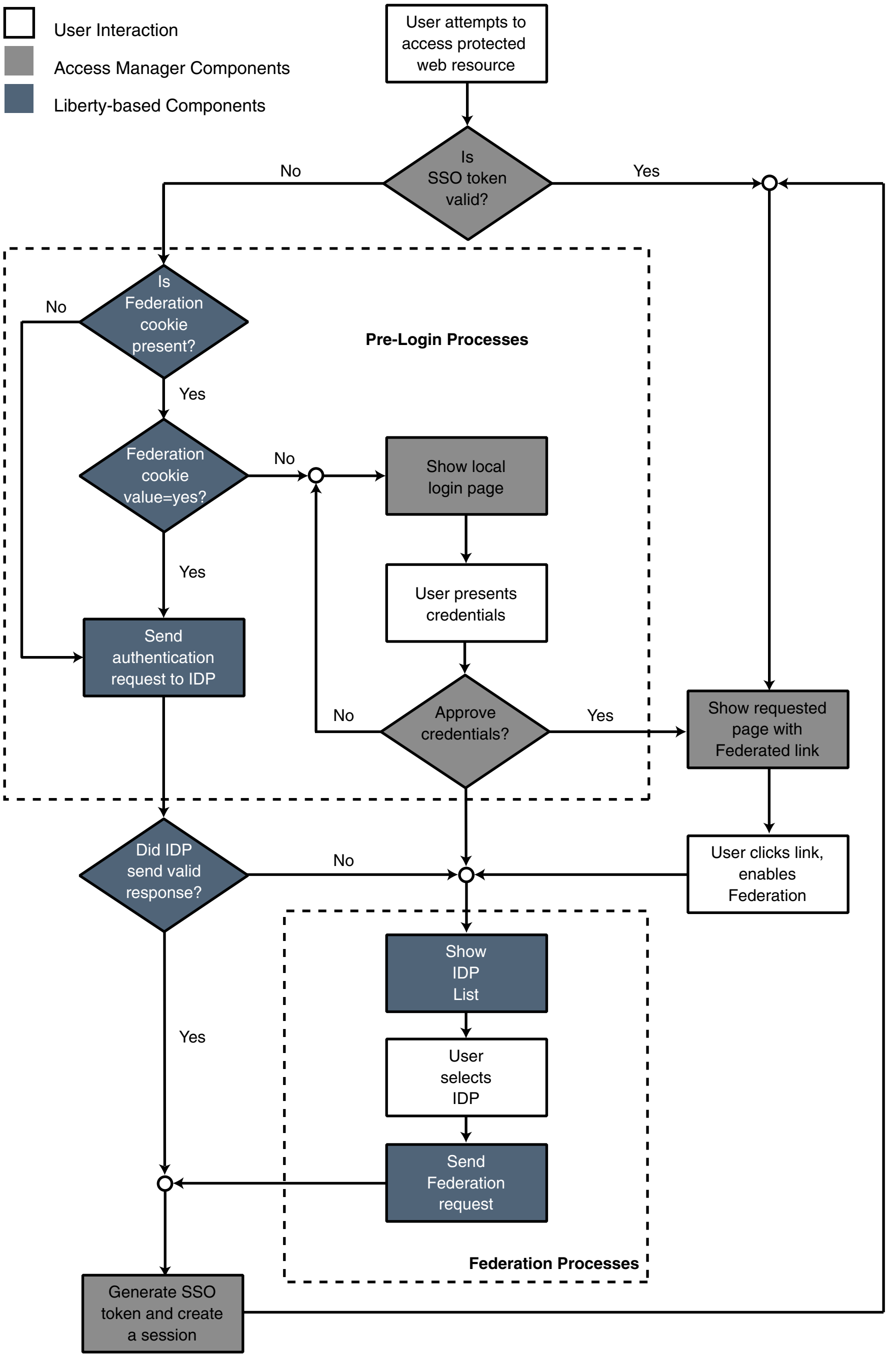

User Interaction
Access Manager Components
Liberty-based Components
User attempts to access protected web resource
Is SSO token valid?
No
Yes
Pre-Login Processes
Is Federation cookie present?
No
Yes
Federation cookie value=yes?
No
Yes
Show local login page
User presents credentials
Send authentication request to IDP
Approve credentials?
No
Yes
Show requested page with Federated link
User clicks link, enables Federation
Did IDP send valid response?
No
Yes
Show IDP List
User selects IDP
Send Federation request
Federation Processes
Generate SSO token and create a session

**FIGURE 11–5** Default Process of Federation

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

---

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

---

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

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

# Using WS-Federation

WS-Federation is part of the larger Web Services Security (WS-Security) framework which provides a means for applying security to web services through the use of security tokens. WS-Security describes how to attach signature and encryption headers as well as security tokens (including binary security tokens such as X.509 certificates and Kerberos tickets) to SOAP messages. WS-Trust, another specification in the WS-Security framework, provides for federation by defining a Security Token Service (STS) and a protocol for requesting and issuing the security tokens. WS-Federation, as implemented in Federated Access Manager, uses the Federated Access Manager Security Token Service (modelled on the WS-Trust specification) to allow providers in different security realms to broker trust using information on identities, identity attributes and authentication, and provider federation. A principal requests a token from the Security Token Services. This token, which may represent the principal's primary identity, a pseudonym, or the appropriate attributes, is presented to the service provider for authentication and authorization. WS-Federation uses several security tokens as well as the mechanism for associating them with messages. This release of Federated Access Manager has implemented the following features of the WS-Federation specification.

- The *Web (Passive) Profile* defines single sign-on, single logout, attribute and pseudonym token exchanges for passive requestors; for example, a web browser that supports HTTP. For the passive mechanisms to provide a single or reduced sign-on, there needs to be a service that will verify that the claimed requestor is really the requestor. Initial verification MUST occur in a secure environment; for example, using SSL/TLS or HTTP/S. The token is abstract and the token exchange is based on the Security Token Service model of WS-Trust.
- Tokens based on the *Web Services-Interoperability Basic Security Profile* (WS-I BSP) define security that is implemented inside a SOAP message; and security implemented at the transport layer, using HTTPS. The protocol covers how you generate or handle security tokens.

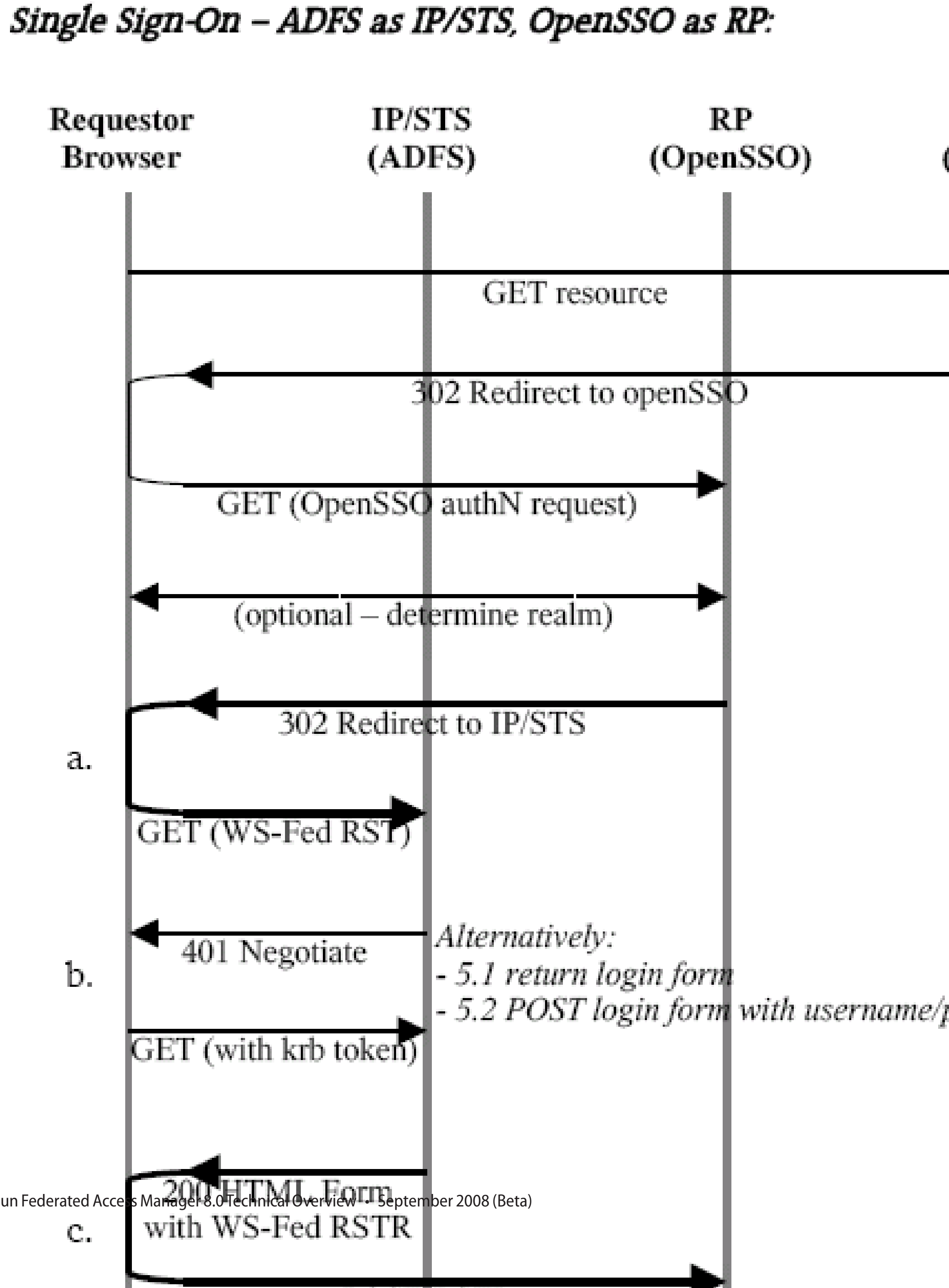
The WS-Federation implementation in Federated Access Manager is based on the application's SAML v2 code and uses WS-Federation 1.1 metadata. Authentication request parameters are represented directly as GET parameters, and the authentication response is a WS-Trust `RequestSecurityTokenResponse` element.

---

**Note –** There is no authentication context mapping, persistent or transient NameID identifiers or auto-federation in the Federated Access Manager implementation of WS-Federation.

---

The entry points for all WS-Federation functionality will be implemented as servlets. JavaServer Pages (JSP) are used only for HTML content (for example, the HTML form used to send the WS-Federation single response from the identity provider to the service provider). The following figure illustrates the flow of messages between Federated Access Manager (acting as the service provider) and the Active Directory (acting as the identity provider).


Single Sign-On – ADFS as IP/STS, OpenSSO as RP:
Requestor Browser
IP/STS (ADFS)
RP (OpenSSO)
GET resource
302 Redirect to openSSO
GET (OpenSSO authN request)
(optional – determine realm)
302 Redirect to IP/STS
a.
GET (WS-Fed RST)
401 Negotiate
b.
GET (with krb token)
Alternatively:
- 5.1 return login form
- 5.2 POST login form with username/
200 HTML Form with WS-Fed RSTR
c.

In a WS-Federation interaction, the request by an unauthenticated user attempting to access a protected web site is redirected to the Active Directory for Federation Services (ADFS) identity provider. After the user is authenticated (either by a back-end single sign-on or by entering credentials), ADFS posts a form containing a signed SAML assertion to the service provider. The service provider validates the assertion, copies the attributes into the user's session, and gives the appropriate access.

---

**Note –** Microsoft Active Directory Federation Services supports single sign-on via WS-Federation.

---

## PART IV

# Delivering Identity Services, Web Services and Web Services Security

This fourth part of the Sun Federated Access Manager Technical Overview contains information on implementing identity services, web services and web services security. It contains the following chapters:

- Chapter 12, "Delivering Identity Web Services"
- Chapter 13, "Accessing and Securing Web Services"

◆ ◆ ◆

## CHAPTER 12

# Delivering Identity Web Services

Federated Access Manager provides client interfaces for authentication, authorization, session, identity management and auditing in Java, in C (C++) and in HTTP(S)/XML. These interfaces are used by web and Java EE policy agents as well as custom applications developed externally. Now, Federated Access Manager also delivers web services that expose these identity functions as simple web services. This chapter contains information on the following topics:

- "About Identity Web Services" on page 171
- "Identity Web Service Styles" on page 172
- "Identity Web Services Architecture" on page 174

## About Identity Web Services

A *web service* is a black-box component that can be accessed using exposed endpoints. Federated Access Manager uses this concept to expose the following security and identity related functions as Identity Web Services:

- authenticate (user credential verification)
- authorize (an authenticated identity's access permission)
- attributes (an authenticated identity's profile)
- log (record and audit actions)

Identity Web Services allow developers to easily invoke these functions without any knowledge of Federated Access Manager, resolving the problems of enabling web service discovery and invocation, security, privacy and ease-of-deployment. Keeping Identity Web Services simple allows an application developer to consume them by pointing an integrated development environment (IDE) to the service's URL and allowing it to generate the stub code that wraps a call to the service. Identity Web Services are supported on:

- NetBeans
- Eclipse
- Visual Studio

---

**Note –** Identity Web Services does not require the Client SDK or deployment of an agent or proxy to protect a resource.

---

Within Identity Web Services the user enters authentication credentials using a JavaServer Pages (JSP). The user data is then forwarded to the composite application which authenticates the web service request. The application may then authorize the operation and obtain the user's profile. See "Identity Web Service Styles" on page 172 for more information.

# Identity Web Service Styles

Federated Access Manager Identity Web Services have been developed in two styles. The decision on which style to use is the initial choice when designing your application. The styles are:

- The SOAP and Web Services Description Language (WSDL) style is the traditional approach preferred by the service-oriented architecture (SOA) business intelligence community. See "SOAP and WSDL" on page 172.
- The REpresentational State Transfer (REST) style is a newer approach preferred by the Web 2.0 community. (A REST service is referred to as *RESTful*.) See "REST" on page 173.

---

**Note –** `developers.sun.com` has an excellent three part article called Securing Applications with Identity Services which contains IDE configuration information and procedures.

---

## SOAP and WSDL

SOAP, WSDL, and XML Schema have become the standard for exchanging XML-based messages among applications. To implement this style, the IDE must obtain the WSDL, generate the client stubs, and set up the JavaServer Pages (JSP) for the Identity Web Services. Once completed, the SOAP Identity Web Services are accessible with the following URLs:

- **http://***host_machine.domain***:8080/opensso/identityservices/IdentityServices**
- **http://***host_machine.domain***:8080/opensso/identityservices?WSDL**

This style may be appropriate when:

- A formal contract must be established to describe the interface that the web service offers. A WSDL is needed to describe the web service interfaces including details such as messages, operations, bindings, and location.
- The architecture must address complex requirements including security, financial transactions, provider trust and the like.

- The architecture needs to handle asynchronous processing and invocation. The infrastructure provided by standards such as WSRM and APIs such as JAX-WS can be leveraged out of the box.

## REST

The internet is comprised of *resources*. Clients may access resources with a URL. When requested, a representation of the resource (an HTML page) is returned. The result of the user clicking a link on the page is that another resource is accessed (possibly an image, video, or another HTML page). Each new representation places the client into a state that is different from the previous state. Thus, the client application changes state with each accessed resource representation. REST is a design architecture in which a web service is viewed as a resource identified by a URL. The web service client then accesses it using a globally defined set of remote methods that describe the action to be performed. REST is not a standard; you can only understand it, and design web services in the REST style. REST does, though, use standards including:

- HTTP
- URLs
- Resource representations (XML, HTML, GIF, JPEG, and others)
- MIME types (text/xml, text/html, image/gif, image/jpeg, and others)

RESTful services are accessed using a generic interface; in Federated Access Manager it is the GET, POST, PUT, and DELETE HTTP methods. The RESTful Identity Web Service is accessible at `http://`*host_machine.domain*`:8080/opensso/identity`. Because these web services are exposed using the HTTP methods, they can be accessed from a browser. This style may be appropriate when:

- The web services are completely stateless. A good test is to consider whether the interaction can survive a restart of the server.
- Bandwidth needs to be limited. REST is particularly useful for limited-profile devices such as PDAs and mobile phones, where the XML payload must be restricted.
- Aggregation into existing web sites is needed. Web services can be exposed with XML and consumed by HTML without significantly reinventing the existing web site architecture.

**Note –** Federated Access Manager REST interfaces currently support only username and password authentication.

# Identity Web Services Architecture

In an Identity Web Service interaction, the user interacts with an application which then calls either of the Identity Web Services to authenticate and authorize the identity, create personalized services, and log the actions. When contacted at the respective URL, Federated Access Manager obtains the user profile from the appropriate identity repository for authentication and the policy configuration from the appropriate configuration data store, and writes the actions to the configured log file. Figure 12–1 illustrates the components of the Identity Web Services.

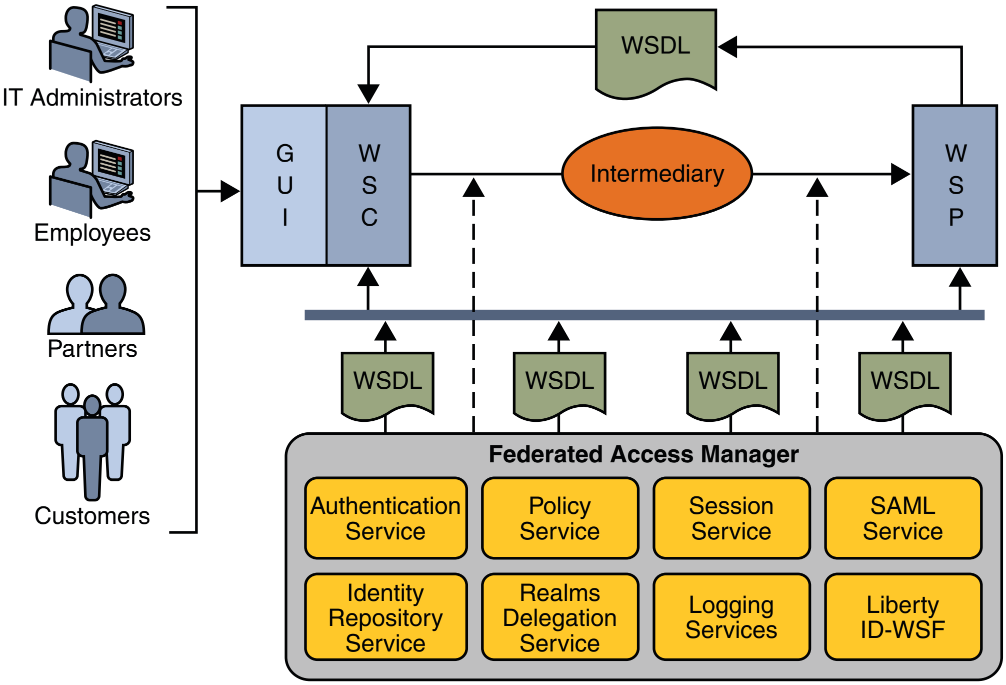

IT Administrators
Employees
Partners
Customers
WSDL
G U I
W S C
Intermediary
W S P
WSDL
WSDL
WSDL
WSDL
Federated Access Manager
Authentication Service
Policy Service
Session Service
SAML Service
Identity Repository Service
Realms Delegation Service
Logging Services
Liberty ID-WSF

**FIGURE 12–1** Components within the Identity Services Interactions

# CHAPTER 13

# Accessing and Securing Web Services

In Federated Access Manager, the Federation framework enables the secure exchange of authentication and authorization information by providing an interface for creating, modifying, and deleting circles of trust and configuring entity providers and identity providers (both remote and hosted types) as entity providers. Additionally, the implemented web services define a stack that supports the Federation framework. This chapter contains general information on the implemented web services in the following sections:

- "About the Web Services Stack" on page 175
- "Web Services Architecture" on page 177
- "Implemented Services" on page 184
- "Web Services Process" on page 184

## About the Web Services Stack

Federated Access Manager fully implements the Liberty Alliance Project Identity-Web Service Framework (Liberty ID-WSF) 1.x specifications and exposes the Web Services Stack as a pluggable framework that can be leveraged for security. Bootstrapping of the framework is accomplished by finding a *resource offering* for an authenticated user using the Discovery Service. Federated Access Manager supports bootstrapping using SAML v2, Liberty ID-FF, or the Authentication Web Service using either of the following modes:

- In browser mode, clients leverage SAML v2 or Liberty ID-FF for bootstrapping.
- In desktop mode, clients leverage the Authentication Web Service for bootstrapping.

Figure 13–1 illustrates the design of the Web Services Stack. Tools and API are provided for web service providers to implement the stack and application developers to consume the services. Federated Access Manager also provides the hooks to quickly integrate an existing infrastructure with a Liberty Alliance Project infrastructure.

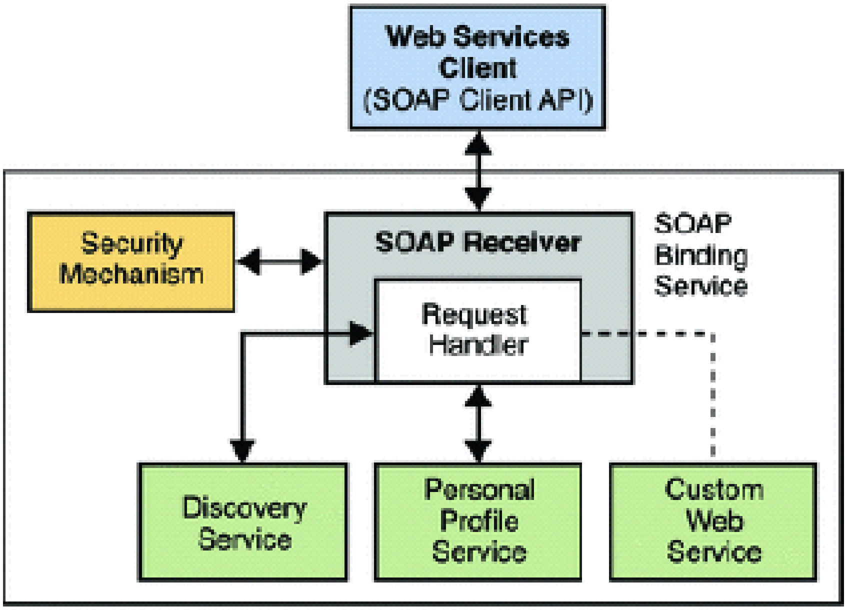

Web Services Client (SOAP Client API)
Security Mechanism
SOAP Receiver
Request Handler
SOAP Binding Service
Discovery Service
Personal Profile Service
Custom Web Service

FIGURE 13–1 Web Services Stack Design

---

**Note –** Any web service developed using the Federated Access Manager Web Services Stack must register with the SOAP Binding Service which provides validation of SOAP messages and generates the `SSOToken` for further authorization.

---

Here is an overview of the interactions between the Web Services Stack and clients.

1. A customer is browsing a merchant site and initiates a purchase.
2. The merchant authenticates the customer, masking the customer's identity and preserving privacy.
3. The merchant uses the Discovery Service to determine the customer's payment service and requests payment on behalf of the principal.
4. The payment service validates the transaction and, before charging the customer, garners user consent using the Interaction Service.
5. With a successful purchase, the merchant returns the confirmation and delivers the item or service.

# Web Services Architecture

Figure 13–2 illustrates the architecture of the web services stack and how a web service client (WSC) communicates with the web service provider (WSP) which, in this case, is Federated Access Manager.

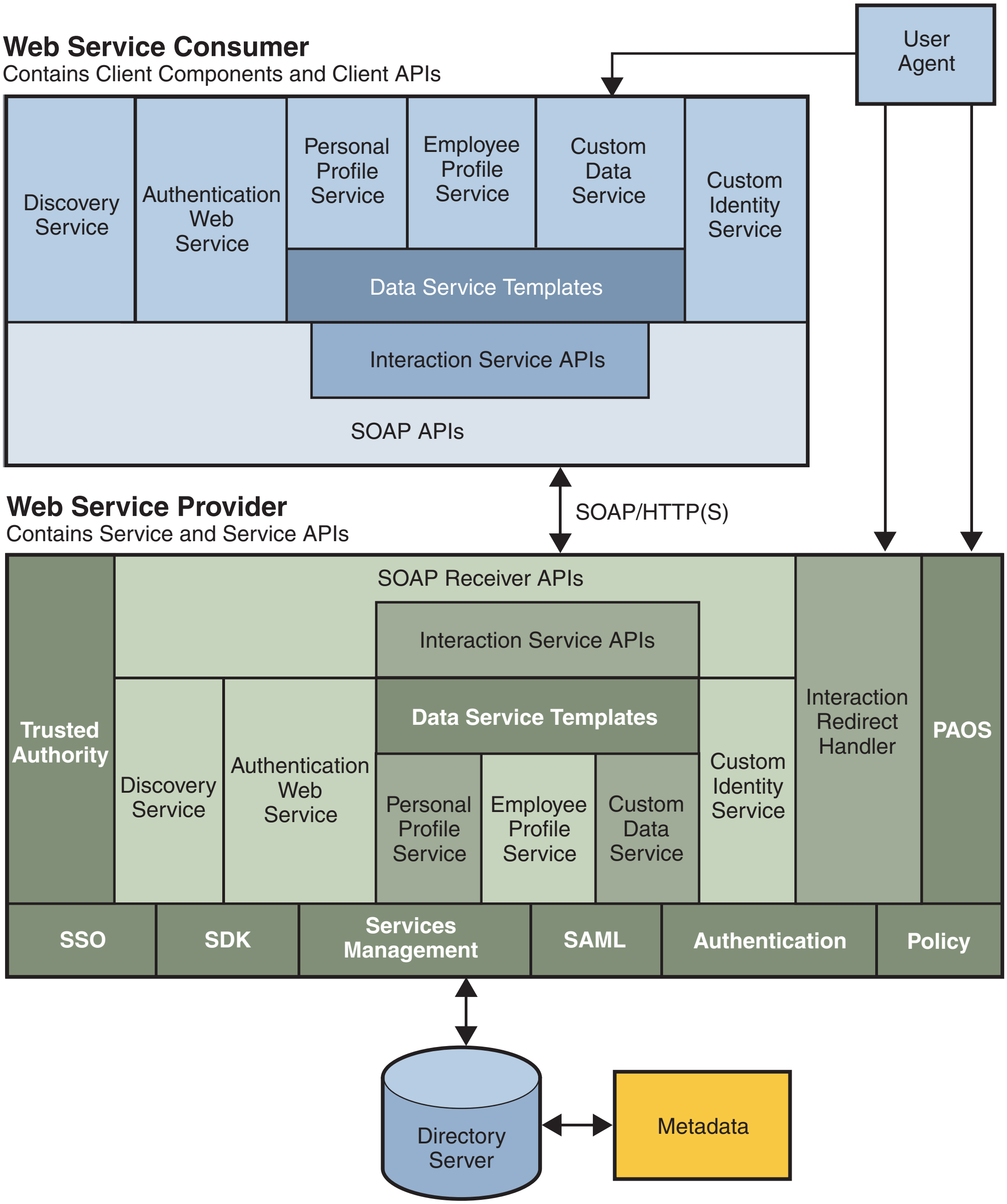

Web Service Consumer
Contains Client Components and Client APIs
User Agent
Discovery Service
Authentication Web Service
Personal Profile Service
Employee Profile Service
Custom Data Service
Custom Identity Service
Data Service Templates
Interaction Service APIs
SOAP APIs
SOAP/HTTP(S)
Web Service Provider
Contains Service and Service APIs
SOAP Receiver APIs
Interaction Service APIs
Trusted Authority
Data Service Templates
Discovery Service
Authentication Web Service
Personal Profile Service
Employee Profile Service
Custom Data Service
Custom Identity Service
Interaction Redirect Handler
PAOS
SSO
SDK
Services Management
SAML
Authentication
Policy
Directory Server
Metadata

FIGURE 13–2 Web Services Architecture

The Web Services Stack can be used in either browser or desktop mode. The following sections describe these modes.

- "Examining Browser Mode Architecture" on page 179
- "Desktop Mode Architecture" on page 182

## Examining Browser Mode Architecture

In browser mode, the service provider (SP) and identity provider (IDP) authenticate an identity using SAML v2. Federated Access Manager can be an SP or an IDP or hosting web service here in this deployment. The SP and the are in the same domain in this deployment. The web service provider registers it's service resource offering with the discovery service before it offers services to various clients. The registration could be done through discovery service protocol or out of band. The Federation Access Manager can be deployed in various roles for this deployment as mentioned in the architecture diagram. The following interaction diagram show cases the typical interaction various entities in this deployment use case.

The browser based clients leverages the SAML2 or ID-FF protocols to bootstrap into IDWSF framework.

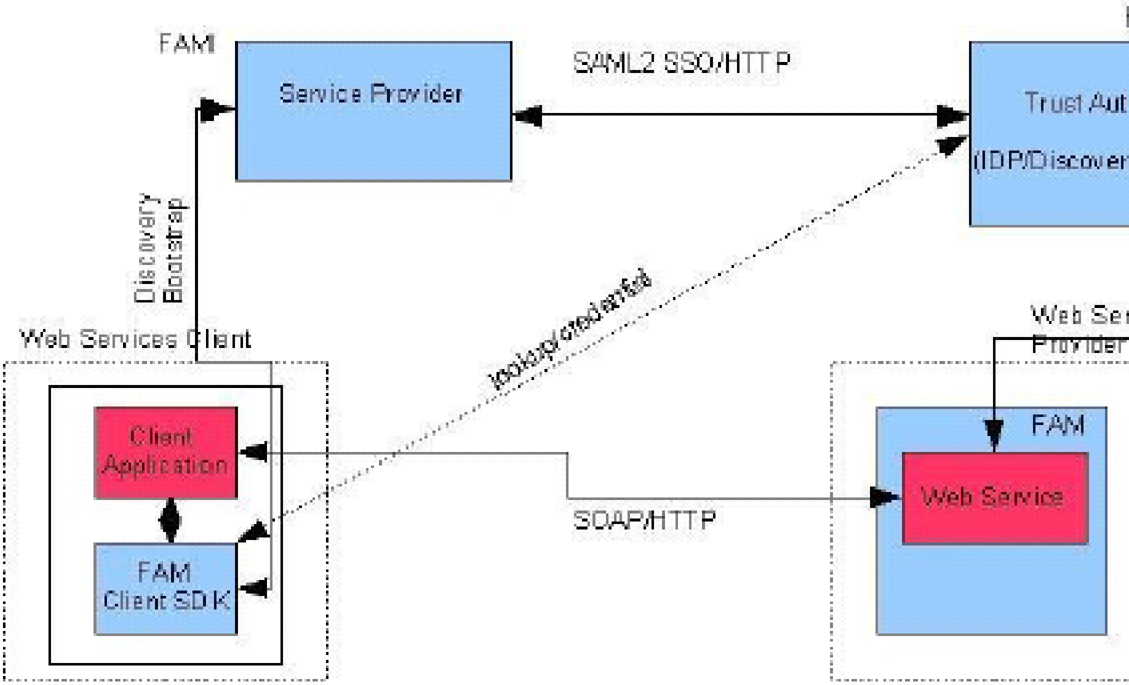

FAM
Service Provider
SAML2 SSO/HTTP
Trust Aut
(IDP/Discover
Discovery
Bootstrap
Web Services Client
lookup credential
Web Se
Provider
Client
Application
FAM
Client SDK
SOAP/HTTP
FAM
Web Service

**FIGURE 13–3** Browser Mode Architecture

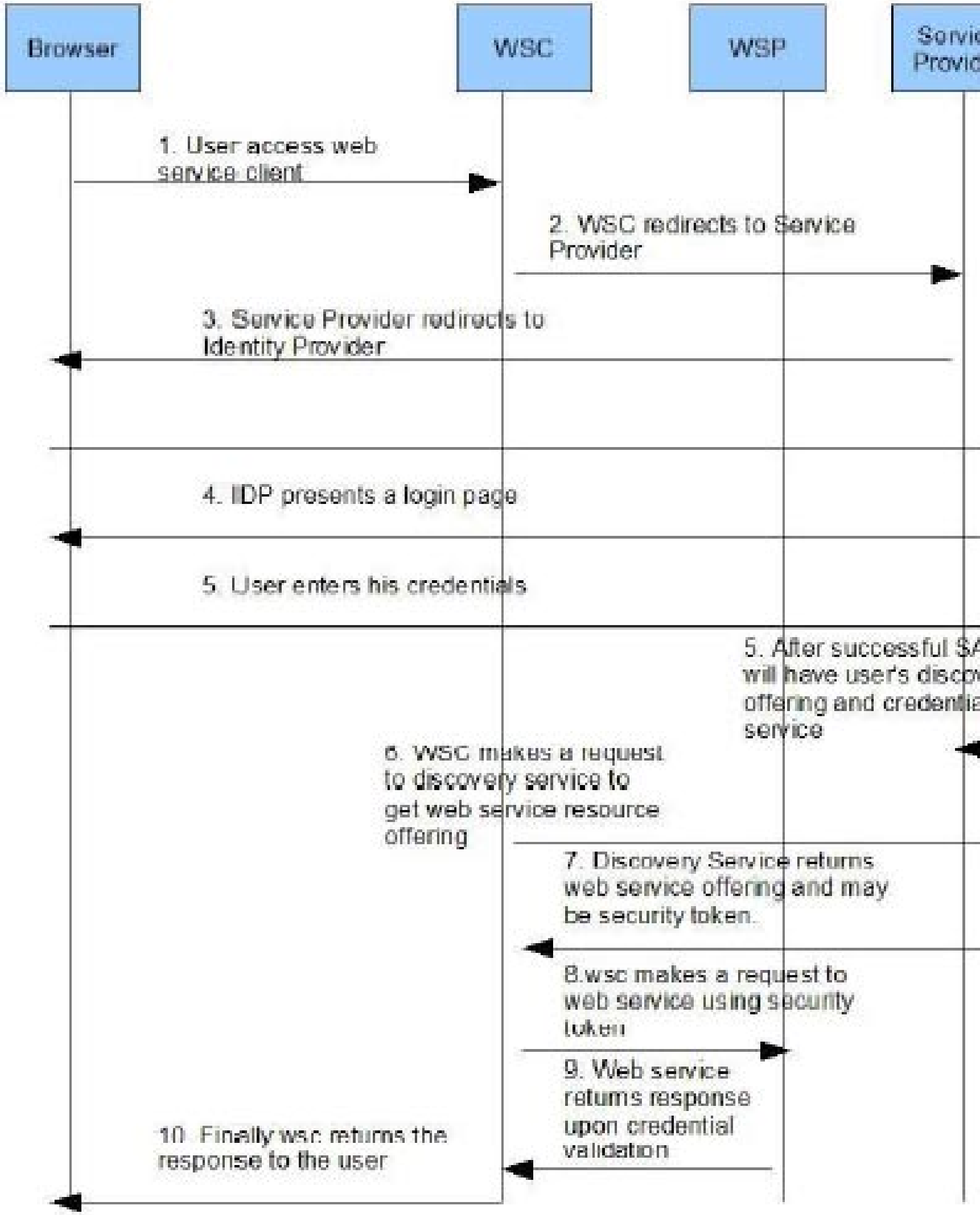

Browser
WSC
WSP
Servic
Provid
1. User access web service client
2. WSC redirects to Service Provider
3. Service Provider redirects to Identity Provider
4. IDP presents a login page
5. User enters his credentials
5. After successful SA
will have user's disco
offering and credentia
service
6. WSC makes a request to discovery service to get web service resource offering
7. Discovery Service returns web service offering and may be security token.
8.wsc makes a request to web service using security token
9. Web service returns response upon credential validation
10. Finally wsc returns the response to the user

**FIGURE 13–4** Browser Mode Flow

## Desktop Mode Architecture

The desktop clients could leverage Liberty Authentication Service to boot strap into ID-WSF. Bootstrapping meaning for ID-WSF is to find the Liberty Discovery Service Resource Offering of an authenticated user.

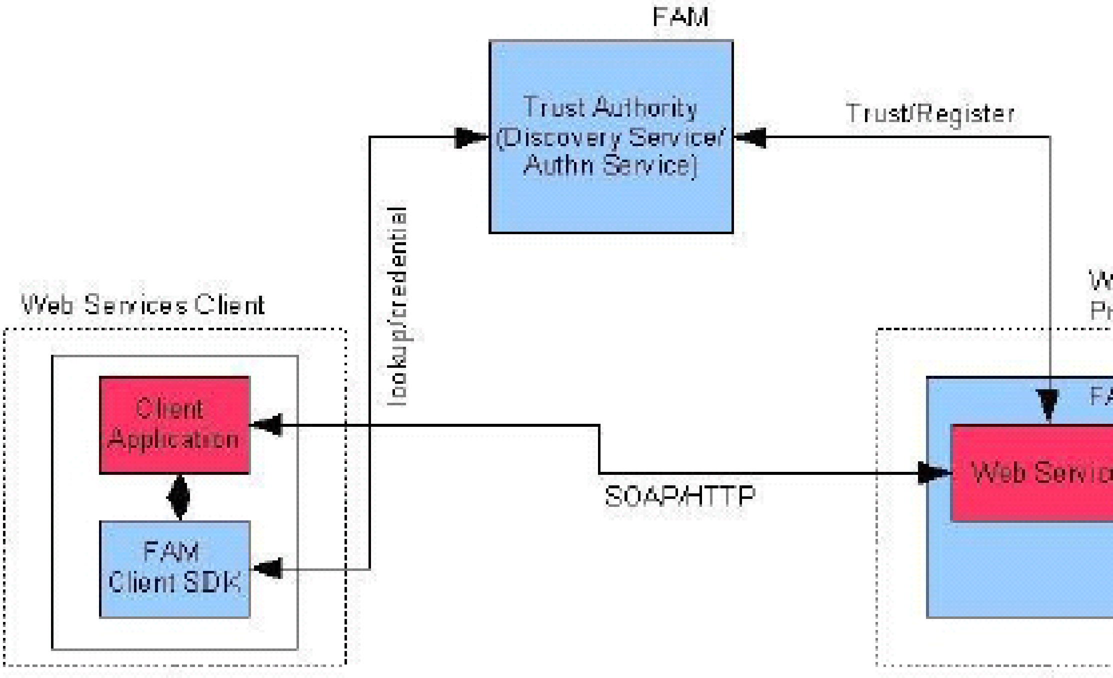

FAM
Trust Authority (Discovery Service/ Authn Service)
Trust/Register
lookup/credential
Web Services Client
Client Application
FAM Client SDK
SOAP/HTTP
W
Pr
FA
Web Servic

**FIGURE 13–5** Desktop Mode Architecture

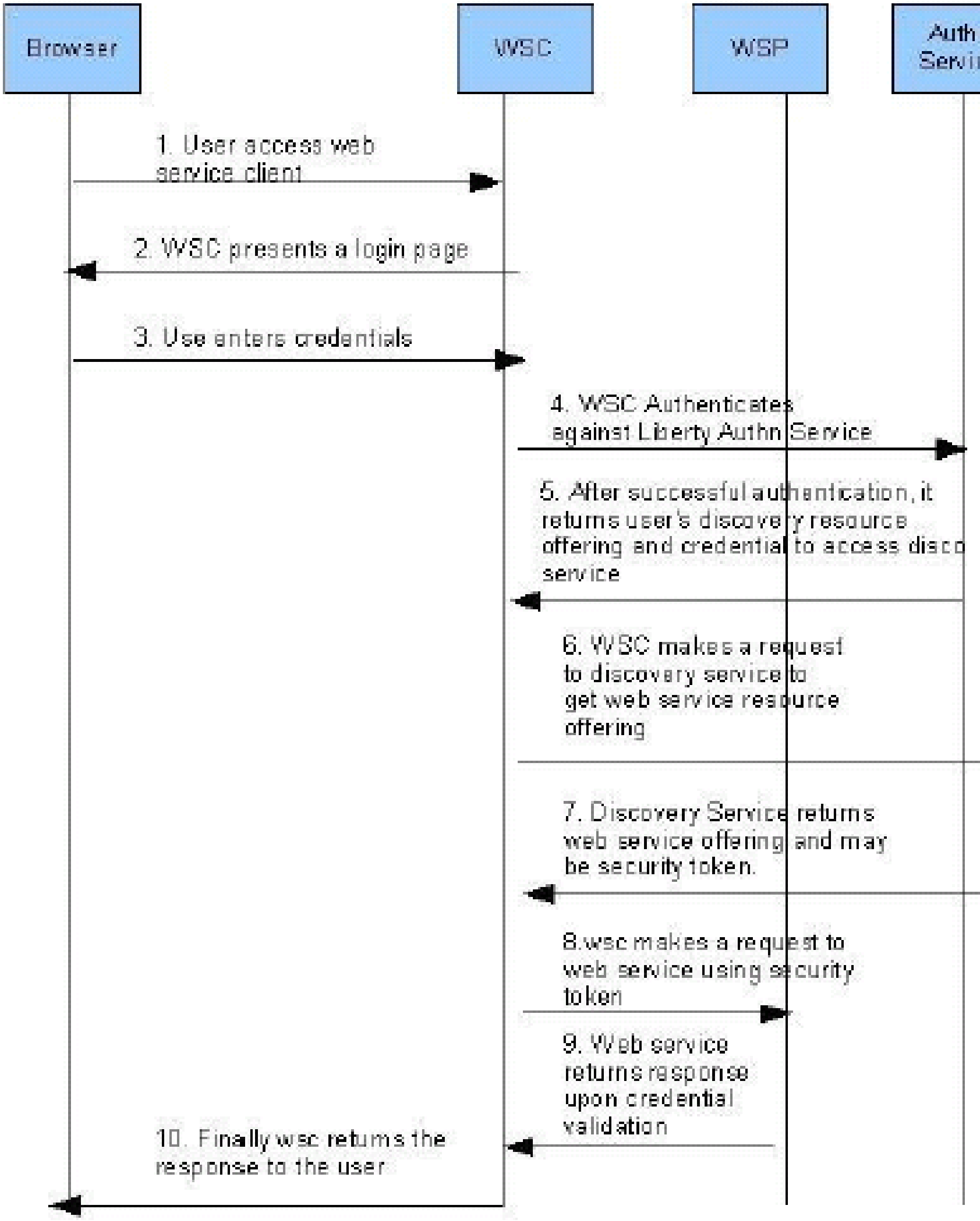

Browser
WSC
WSP
Auth Servi
1. User access web service client
2. WSC presents a login page
3. Use enters credentials
4. WSC Authenticates against Liberty Authn Service
5. After successful authentication, it returns user's discovery resource offering and credential to access disco service
6. WSC makes a request to discovery service to get web service resource offering
7. Discovery Service returns web service offering and may be security token.
8.wsc makes a request to web service using security token
9. Web service returns response upon credential validation
10. Finally wsc returns the response to the user

**FIGURE 13–6** Desktop Mode Flow

# Implemented Services

Federated Access Manager includes the following web services:

Authentication Web Service
Provides authentication to a WSC, allowing the WSC to obtain security tokens for further interactions with other services at the same provider. Upon successful authentication, the final Simple Authentication and Security Layer (SASL) response contains the resource offering for the Discovery Service.

Discovery Service
A web service that allows a requesting entity, such as a service provider, to dynamically determine a principal's registered attribute provider. Typically, a service provider queries the Discovery Service, which responds by providing a *resource offering* that describes the requested attribute provider. The implementation of the Discovery Service includes Java and web-based interfaces.

SOAP Binding
A set of Java APIs used by the developer of a Liberty-enabled identity service. The APIs are used to send and receive identity-based messages using SOAP, an XML-based messaging protocol.

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

# Web Services Process

The following figure provides a high-level view of the process between the various components in the web services stack. In this example:

- The web browser represents a user.
- The service provider also acts as a web services consumer (WSC), invoking a web service on behalf of the user. The service provider relies on the identity provider for authentication.
- The identity provider acts as an authentication provider by authenticating the user. It also acts as a trusted authority, issuing security tokens through the Discovery Service.
- The web services provider (WSP) serves requests from web services clients such as the Liberty Personal Profile Service.

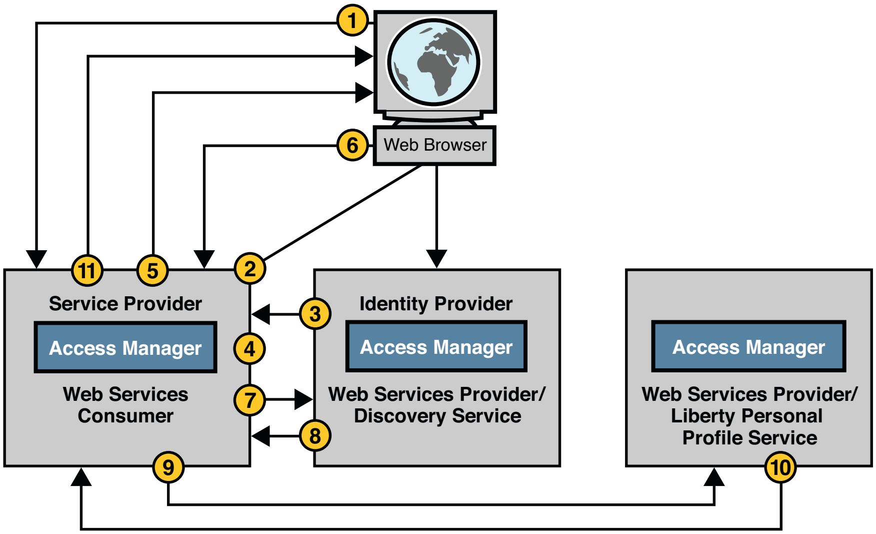

1
Web Browser
6
11
5
2
Service Provider
Access Manager
Web Services Consumer
3
4
7
8
9
Identity Provider
Access Manager
Web Services Provider/ Discovery Service
Access Manager
Web Services Provider/ Liberty Personal Profile Service
10

FIGURE 13–7 Web Services Stack Process

The following process assume that the user, the identity provider, and the service provider have already been federated.

1. The user attempts to access a resource hosted on the service provider server.
2. The service provider redirects the user to the identity provider for authentication.
3. The identity provider authenticates the user successfully and sends the single sign-on assertion to the requesting service provider.
4. The service provider verifies the assertion and the user is issued a session token.
5. The service provider redirects the user to the requested resource.
6. The user requests access to another service hosted on the WSC server.

   For example, it might need that value of an attribute from the user's Liberty Personal Profile Service.
7. The WSC sends a query to the Discovery Service to determine where the user's Liberty Personal Profile Service instance is hosted.

   The WSC bootstraps the Discovery Service with the resource offering from the assertion obtained earlier.
8. The Discovery Service returns a response to the WSC containing the endpoint for the user's Liberty Personal Profile Service instance and a security token that the WSC can use to access it.
9. The WSC sends a query to the Liberty 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 Liberty Personal Profile Service resource offering must be followed.

10. The Liberty Personal Profile Service instance authenticates and validates authorization for the requested user or the WSC, or 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 Liberty Personal Profile Service instance returns a response to the WSC after collecting all required data.
11. The WSC processes the Liberty Personal Profile Service response, and renders the service pages containing the information.

For detailed information about all these components, see the XXXXXX Sun Java System Access Manager 7.1 Federation and SAML Administration Guide.

## PART V

# Additional Features

This final section of the Sun Federated Access Manager Technical Overview contains information on the Logging Service and third-party product information. It contains the following chapters:

- Chapter 14, "Logging and the Java Enterprise System Monitoring Framework"

# CHAPTER 14

# Logging and the Java Enterprise System Monitoring Framework

Sun Federated Access Manager provides its own logging feature that records information such as user login, user logout, session creation, and policy evaluation. This chapter describes how Access Manager logging works, and provides some information about the Java Enterprise System Monitoring Framework. It contains the following sections:

**Caution –** **See common criteria logging paper by Burt Fujii**

- "Logging Overview" on page 189
- "Log Files" on page 191
- "Access Manager Component Logs" on page 194
- "Additional Logging Features" on page 195
- "Java Enterprise System Monitoring Framework" on page 196

## Logging Overview

The Logging Service enables Access Manager services to record information such as access denials, access approvals, authentication events, and authorization violations. Administrators can use the logs to track user actions, analyze traffic patterns, audit system usage, review authorization violations, and troubleshoot. The logged information from all Access Manager services is recorded in one centralized directory. The default location for all Access Manager log files is `/var/opt/SUNWam/logs`. Logging client APIs enable external applications to access the Logging framework. This section contains the following:

- "Logging Service" on page 190
- "Logging Configuration" on page 190
- "Recorded Events" on page 190

# Logging Service

The Logging Service stores the attributes and values for the logging function. A global service configuration file named `amLogging.xml` defines the Logging attributes. Examples of Logging Service attributes are maximum log size, log location, and log format (flat file or relational database). The attribute values are applied across the Access Manager deployment and inherited by every configured realm. By default, `amLogging.xml` is located in the directory `/etc/opt/SUNWam/config/xml` when Access Manager is installed in a Solaris environment. (When installed on Windows, the directory is *jes-install-dir*`\identity\config\xml`; on HP-UX the directory is `/etc/opt/sun/identity/config/xml`.) The structure of `amLogging.xml` is defined by file `sms.dtd`.

# Logging Configuration

When Access Manager starts or when any logging configuration data is changed using the Access Manager console, the logging configuration data is loaded (or reloaded) into the Logging Service. This data includes the log message format, log file name, maximum log size, and the number of history files. Applications can use the client APIs to access the Logging features from a local or remote server. The client APIs use an XML-over-HTTP layer to send logging requests to the Logging component on the server where Access Manager is installed.

# Recorded Events

The client passes the Logging Service logs information to the `com.sun.identity.log.LogRecord` class. The following table summarizes the items logged by default in the `LogRecord`.

**TABLE 14–1** Events Recorded in LogRecord

| Event | Description |
|---|---|
| Time | The date (`YYYY-MM-DD`) and time (`HH:MM:SS`) at which the log message was recorded. This field is not configurable. |
| Data | Variable data pertaining to the log records's `MESSAGE ID`. This field is not configurable. |
| Module Name | Name of the Access Manager service or application being logged. Additional information on the value of this field can be found in “Adding Log Data” on page 88. |
| Domain | Access Manager domain to which the user belongs. |
| Log Level | The Java 2 Platform, Standard Edition (J2SE) version 1.4 log level of the log record. |

**TABLE 14–1** Events Recorded in LogRecord *(Continued)*

| Event | Description |
|---|---|
| Login ID | ID of the user as the subject of the log record. The user ID is taken from the session token. |
| IP Address | IP address from which the operation was performed. |
| Logged By | User who writes the log record. The information is taken from the session token passed during `logger.log(logRecord, ssoToken)`. |
| Host Name | Host name associated with the IP Address above. |
| MessageID | Non-internationalized message identifier for this log record's message. |
| ContextID | Identifier associated with a particular login session. |

# Log Files

The following sections contain information about Access Manager log files:

- "Log File Formats" on page 191
- "Error and Access Logs" on page 193

## Log File Formats

Access Manager can record events in either of the following formats:

- "Flat File Format" on page 191
- "Relational Database Format" on page 192

### Flat File Format

The default flat file format is the W3C Extended Log Format (ELF). Access Manager uses this format to record the default fields in each log record. See "Recorded Events" on page 190 for a list of default fields and their descriptions. The following example illustrates an authentication log record formatted for a flat file. The fields are in this order: `Time`, `Data`, `ModuleName`, `MessageID`, `Domain`, `ContextID`, `LogLevel`, `LoginID`, `IPAddr`, `LoggedBy`, and `HostName`.

**EXAMPLE 14–1** Flat File Record From `amAuthentication.access`

```
"2005-08-01 16:20:28"   "Login Success" LDAP    AUTHENTICATION-100
   dc=example,dc=com       e7aac4e717dda1bd01      INFO
uid=amAdmin,ou=People,dc=example,dc=com 192.18.187.152
"cn=exampleuser,ou=Example Users,dc=example,dc=com" exampleHost
```

## Relational Database Format

When Access Manager uses a relational database to log messages, the messages are stored in a database table. Access Manager uses Java Database Connectivity (JDBC) to access the database table. JDBC provides connectivity to a wide range of SQL databases. JDBC also provides access to other tabular data sources such as spreadsheets or flat files. Oracle® and MySQL databases are currently supported.

For log records generated by Access Manager, the `Data` and `MessageID` fields are used slightly differently than in previous versions of Access Manager. Starting with this version of Access Manager, the `MessageID` field is introduced as a template for types of log messages. For example, in previous versions, Access Manager would generate the following message in the `Data` field:

```
Data: "Created group
cn=agroupSubscription1,ou=Groups,dc=iplanet,dc=com"
```

In this version of Access Manager, two log records are recorded for the one event:

```
Data:       agroupSubscription1|group|/
MessageID:    CONSOLE-1
```

and

```
Data:       agroupSubscription1|group|/
MessageID:    CONSOLE-2
```

These log records reflect the use of identities and realms. In this example, `CONSOLE-1` indicates an attempt to create an identity object, and `CONSOLE-2` indicates the attempt to create an identity object was successful. The root organization notation (`dc=iplanet,dc=com`) is replaced with a forward slash (`/`). The variable parts of the messages (*agroupSubscription1*, *group*, and */*) are separated by a pipe character (|), and continue to go into the `Data` field of each log record. The `MessagID` string is not internationalized in order to facilitate machine-readable analysis of the log records in any locale.

The following table summarizes the schema for a relational database.

**TABLE 14–2** Relational Database Log Format

| Column Name | Data Type | Description |
|---|---|---|
| `TIME` | VARCHAR(30) | Date of the log in the format `YYYY-MM-DD HH:MM:SS`. |
| `DATA` | VARCHAR(1024) | The variable data part of the log record pertaining to the MESSAGE ID. For MySQL, the Data Type is VARCHAR(255). |
| `MODULENAME` | VARCHAR(255) | Name of the Access Manager component invoking the log record. |

**TABLE 14–2** Relational Database Log Format *(Continued)*

| Column Name | Data Type | Description |
|---|---|---|
| `DOMAIN` | VARCHAR(255) | Access Manager domain of the user. |
| `LOGLEVEL` | VARCHAR(255) | JDK 1.4 log level of the log record. |
| `LOGINID` | VARCHAR(255) | Login ID of the user who performed the logged operation. |
| `IPADDR` | VARCHAR(255) | IP Address of the machine from which the logged operation was performed. |
| `LOGGEDBY` | VARCHAR(255) | Login ID of the user who writes the log record. |
| `HOSTNAME` | VARCHAR(255) | Host name of machine from which the logged operation was performed. |
| `MESSAGE ID` | VARCHAR(255) | Non-internationalized message identifier for this log record's message. |
| `CONTEXT ID` | VARCHAR(255) | Identifier associated with a particular login session. |

## Error and Access Logs

There are two types of Access Manager log files:

- Access log files
- Error log files

*Access log files* record general auditing information concerning the Access Manager deployment. An access log may contain a single record for an event such as a successful authentication, or multiple records for the same event. For example, when an administrator uses the console to change an attribute value, the Logging Service logs the attempt to change in one record but, the Logging Service also logs the results of the execution of the change in a second record. *Error log files* record errors that occur within the application. While an operation error is recorded in the error log, the operation attempt is recorded in the access log file.

Flat log files are appended with the `.error` or `.access` extension. Database column names end with `_ERROR` or `_ACCESS`. For example, a flat file logging console events is named `amConsole.access` while a database column logging the same events is named `AMCONSOLE_ACCESS` or `amConsole_access`.

---

**Note –** The period (.) separator in a log filename is converted to an underscore (_) in database formats. Also in databases, table names may be converted to all upper case. For example, `amConsole.access` may be converted to `AMCONSOLE_ACCESS`, or it may be converted to `amConsole_access`.

---

# Access Manager Component Logs

The log files record a number of events for each of the Access Manager components using the Logging Service. Administrators typically review these log files on a regular basis. The default location for all Access Manager log files is `/var/opt/SUNWam/logs` when Access Manager is installed in a Solaris environment. (When installed on Windows, the directory is *jes-install-dir*`\identity\logs`; on HP-UX the directory is `/var/opt/sun/identity/logs`.) The following table provides a brief description of the log files produced by each Access Manager component.

**TABLE 14–3** Access Manager Component Logs

| Component | Log Filename | Information Logged |
|---|---|---|
| Session | ■ `amSSO.access` | Session management attributes values such as login time, logout time, and time out limits. Also session creations and terminations. |
| Administration Console | ■ `amConsole.access`<br>■ `amConsole.error` | User actions performed through the administration console such as creation, deletion and modification of identity-related objects, realms, and policies. `amConsole.access` logs successful console events while `amConsole.error` logs error events. |
| Authentication | ■ `amAuthentication.access`<br>■ `amAuthentication.error` | User logins and log outs, both successful and failed. |
| Federation | ■ `amFederation.access`<br>■ `amFederation.error`<br>■ `amLiberty.access`<br>■ `amLiberty.error` | Federation-related events such as the creation of an authentication domain or the creation of a hosted provider entity. |
| Authorization (Policy) | ■ `amPolicy.access`<br>■ `amPolicy.error`<br>■ `amAuthLog` | Policy-related events such as policy creation, deletion, or modification, and policy evaluation. `amPolicy.access` logs policy allows, `amPolicy.error` logs policy error events, and `amAuthLog` logs policy denies. |
| Policy Agent | `amAgent` | Exceptions regarding resources that were either accessed by a user or denied access to a user. `amAgent` logs reside on the server where the policy agent is installed. Agent events are logged on the Access Manager machine in the Authentication logs. |
| SAML | ■ `amSAML.access`<br>■ `amSAML.error` | SAML-related events such as assertion and artifact creation or removal, response and request details, and SOAP errors. |

**TABLE 14–3** Access Manager Component Logs *(Continued)*

| Component | Log Filename | Information Logged |
|---|---|---|
| Command-line | `amAdmin.access`<br>`amAdmin.error` | Event successes and errors that occur during operations using the command line tools. Examples are: loading a service schema, creating policy, and deleting users. |
| Password Reset | `amPasswordReset.access` | Password reset events. |

For detailed reference information about events recorded in each type of Access Manager log, see the *Sun Java System Access Manager 7.1 Administration Guide*.

# Additional Logging Features

You can enable a number of logging features for added functionality. The additional features include:

- "Secure Logging" on page 195
- "Remote Logging" on page 195
- "Log Reading" on page 196

## Secure Logging

This feature adds an extra measure of security to the Logging Service. When secure logging is enabled, the Logging component can detect unauthorized changes to the security logs. No special coding is required to leverage this feature. However, secure logging uses a certificate that you must create and install in the container that runs Access Manager. When secure logging is enabled, a Manifest Analysis and Certification (MAC) is generated and stored for every log record, and a special signature record is periodically inserted in the log. The signature record represents the signature for the contents of the log written up to that point. The combination of the certificate and the signature record ensures that the logs have not been tampered. For detailed information about enabling secure logging, see the *Sun Java System Access Manager 7.1 Administration Guide*.

## Remote Logging

Remote logging allows a client using the Client APIs to create log records on an instance of Access Manager deployed on a remote machine. Remote logging is useful in the following situations:

- When the login URL in the Naming Service of an Access Manager instance points to a remote Access Manager instance, and a trust relationship between the two instances has been configured.

- When the Access Manager APIs are installed in a remote Access Manager instance, and a client application or a simple Java class running on the Access Manager server uses the logging APIs.
- When logging APIs are used by Access Manager agents.

## Log Reading

Access Manager provides Logging APIs for writing your own custom log reading program. You can set up queries to retrieve specific records from the log file or database. This is useful for auditing purposes. For more information, see the *Sun Java System Access Manager 7.1 Developer's Guide*.

# Java Enterprise System Monitoring Framework

Access Manager 7.1 integrates with the Java Enterprise System (JES) monitoring framework through Java Management Extensions (JMX). JMX technology provides the tools for building distributed, web-based, modular, and dynamic solutions for managing and monitoring devices, applications, and service-driven networks. Typical uses of the JMX technology include: consulting and changing application configuration, accumulating statistics about application behavior, notification of state changes and erroneous behaviors. Data is delivered to centralized monitoring console. Access Manager 7.1 uses the Java ES Monitoring Framework to capture statistics and service-related data such as:

- Number of attempted, successful, and failed authentications
- Number of active sessions, statistics from session failover DB
- Session failover database statistics
- Policy caching statistics
- Policy evaluation transaction times
- Number of assertions for a given provider in a SAML/Federation deployment

For comprehensive information about how the JES monitoring framework works and how you can use the monitoring framework with Access Manager, see the *Sun Java Enterprise System 5 Update 1 Monitoring Guide*.

# Index

**A**

access control, Federated Access Manager, 21-22
access logs, 193-194
Access Manager, legacy mode, 54
Access Manager Repository Plug-in, identity repository plug-in, 59-60
account locking
  and authentication, 102-103
  memory locking, 102
  physical locking, 102
action, normal policy, 119
Active Directory authentication, 106
active session time, policy, 121
administration tools, description, 63
agent, *See* policy agents
agents
  *See* authentication agent
  *See* policy agent
`amLogging.xml`, 190
Anonymous authentication, 106
API, SAML v2, 151
application programming interfaces, *See* API
architecture
  Federated Access Manager, 27-29
  federation, 142-143
  plug-ins layer, 63-64
  SAML v1.x, 152-153
auditing, *See* logging
authentication agent, overview, 62
authentication chain, policy, 121
authentication chaining, 103-104
authentication configuration service, 111
authentication context, overview, 157-159
authentication data, 54-60, 60
authentication level, policy, 121
authentication level-based authentication, 109
authentication module instance, policy, 121
authentication modules, 106-108
  Active Directory, 106
  Anonymous, 106
  Certificate, 106
  Data Store, 106
  Federation, 107
  HTTP Basic, 107
  JDBC, 107
  Membership, 107
  MSISDN, 107
  RADIUS, 107
  SafeWord, 107
  SAML, 107
  SecurID, 107
  UNIX, 107
  Windows Desktop SSO, 108
  Windows NT, 108
Authentication Service
  account locking, 102-103
  authentication chaining, 103-104
  authentication configuration service, 111
  authentication level-based authentication, 109
  authentication type configurations, 108-110
  client detection, 102
  configuration, 110-111
  core authentication module, 110
  description, 32-34

Authentication Service *(Continued)*
  distributed authentication user interface, 113-114
  features, 102-106
  FQDN name mapping, 104
  JAAS shared state, 105
  login URLs, 111
  module-based authentication, 109
  modules, 106-108
  organization-based authentication, 109
  overview, 99-101
  persistent cookie, 104
  process, 85-87
  realm-based authentication, 108
  realm configuration, 110
  redirection URLs, 111
  role-based authentication, 108
  service-based authentication, 109
  session upgrade, 105
  SPI, 64
  user-based authentication, 109
  user interface, 111-113
authentication type configurations, 108-110
Authentication Web Service, 184
  description, 46
authorization
  *See* Policy Service
  and XACML, 123-127
  overview, 117-118
auto-federation, 156

## B

basic user session, 83-93
  initial HTTP request, 83-85
`bootstrap`, 60-61
bulk federation, 156

## C

CDSSO, *See* cross-domain single sign-on
centralized configuration data, `bootstrap`, 60-61
Certificate authentication, 106
circle of trust
  definition, 132-133, 133-135
client detection, and authentication, 102
Client Detection Service, in authentication, 85-87
Client SDK, description, 63
common domain, 159-160
  reader service, 160
  writer service, 160
common domain cookie, 160
conditions, policy, 121
configuration, Authentication Service, 110-111
configuration data, 54-60
configuration data store, `bootstrap`, 60-61
cookies
  and sessions, 81-82
  common domain, 160
core authentication module, 110
cross-domain single sign-on
  definition, 37-40, 80
  process, 95-97
current session properties, policy, 121

## D

data
  authentication, 60
  configuration, 54-57
  identity, 57-60
  types of, 54-60
Data Store authentication, 106
data stores, 54-60
definitions
  circle of trust, 133-135
  federation, 131-133
  identity, 131-132
  identity federation, 131-132
  identity provider, 133-135
  principal, 133-135
  provider, 133-135
  provider federation, 132-133
  service provider, 133-135
  trust, 133
Discovery Service, 184
  description, 46

distributed authentication
  definition, 113-114
  in authentication, 85-87
documentation, 12-13
  adjunct products, 13
DTD
  files used, 46-47
  modifying files, 46-47
dynamic identity provider proxying, Liberty ID-FF, 161

**E**

error logs, 193-194

**F**

failover, configuration data store, 60-61
`famadm`, description, 63
`famAdminTools.zip`, description, 63
`famSessionTools.zip`, description, 63
features, Authentication Service, 102-106
Federated Access Manager
  access control, 21-22
  architecture, 27-29
  configuration data, 54-57
  data stores, 54-60
  federation management, 22
  functions, 21-23
  identity data, 57-60
  identity services, 23
  infrastructure, 54-64
  introduction, 19-20
  legacy mode, 54
  overview, 20
  process, 29-31
  services, 31-50
  web services security, 22-23
federated identity, 131-132
federation, 133-135, 145-167
  architecture, 142-143
  common domain, 159-160
  definition, 131-133
federation *(Continued)*
  identity federation and single sign-on, 155-156
  SPI, 64
Federation authentication, 107
federation management, Federated Access Manager, 22
federation options
  Liberty ID-FF, 145-146
  SAML v1.x, 145-146, 146-154
  SAML v2, 145-146, 146-154
Federation Services, description, 44-46
flat files, logging, 191
FQDN name mapping, and authentication, 104
functions, Federated Access Manager, 21-23

**G**

General Policy Service, 117-118
global logout, Liberty ID-FF, 161

**H**

HTTP Basic authentication, 107
HTTP request, and authentication, 83-85

**I**

identifiers, Liberty ID-FF, 160-161
identity, definition, 131-132
identity data, 54-60
identity federation, 133-135, 155-156
  definition, 131-132
identity providers, definition, 133-135
Identity Repository Service
  *See* identity data
  description, 42-44
identity repository service, plug-in, 64
identity services, Federated Access Manager, 23
information tree, *See* configuration data
infrastructure, Federated Access Manager, 54-64
introduction, Federated Access Manager, 19-20
IP address/DNS names, policy, 121

**J**

JAAS framework, and authentication, 103-104
JAAS shared state, and authentication, 105
JavaServer Pages, *See* JSP
JDBC, 192-193
JDBC authentication, 107
JSP, SAML v2, 152

**L**

LDAP authentication, 107
LDAP filter, policy, 121
LDAPv3, identity repository plug-in, 59
legacy mode, Federated Access Manager, 54
Liberty Alliance Project Identity Federation Framework, *See* Liberty ID-FF
Liberty ID-FF, 154-164
    and single sign-on, 161-164
    auto-federation, 156
    bulk federation, 156
    convergence with SAML, 148-152
    dynamic identity provider proxying, 161
    federation option, 145-146
    global logout, 161
    identifiers and name registration, 160-161
    pre-login process, 161-164
    process, 161-164
    SAML v1.x comparison, 145-146
Liberty Personal Profile Service, 184
    description, 46
local identity, 131-132
log reading, 196
logging
    access logs, 193-194
    `amLogging.xmll`, 190
    component log filenames, 194
    error logs, 193-194
    flat files, 191
    log reading, 196
    overview, 189-191
    process, 91-93
    recorded events, 190-191
    relational databases, 192-193
    remote logging, 195-196
logging *(Continued)*
    secure logging, 195
Logging Service, description, 40-42
login URLs, and authentication, 111

**M**

Membership authentication, 107
memory locking, and authentication, 102
module-based authentication, 109
MSISDN authentication, 107

**N**

name registration, Liberty ID-FF, 160-161
Naming Service, and session validation, 87-89
normal policy
    condition, 121
    policy types, 119-122
    rule, 119
    subject, 119-120

**O**

organization-based authentication, 109
overview
    *See* authentication agent
    *See* policy agent
    authentication and authentication context, 157-159
    Authentication Service, 99-101
    Federated Access Manager, 20
    Policy Service, 117-118
    session service, 79-80
    XACML, 123-127

**P**

PDP, in SAML, 152
persistent cookie, and authentication, 104
physical locking, and authentication, 102

plug-ins
  Access Manager Repository Plug-in, 59-60
  architecture, 63-64
  authentication
    *See* authentication modules
  identity repository service, 64
  LDAPv3, 59
  policy response providers, 122
  Policy Service, 64
  service configuration, 64
policy
  and XACML, 123-127
  conditions, 121
  definition, 117-118
  General Policy Service, 117-118
  Policy Configuration Service, 117-118
  rule, 119
  subject, 119-120
Policy Administration Point, definition, 117-118
policy agent, overview, 61-62
policy agents, 35-37
Policy Configuration Service, 117-118
Policy Decision Point
  and XACML, 123-127
  definition, 117-118
Policy Enforcement Point
  and XACML, 123-127
  definition, 117-118
policy evaluation, process, 89-91
Policy Service, 35-37
  definition, 117-127
  description, 35-37
  normal policy, 119-122
  overview, 117-118
  plug-in, 64
  policy evaluation, 89-91
  policy response provider plug-in, 122
  referral policy, 122
  XACML, 123-127
policy types, 119-122
  normal policy, 119-122
  referral policy, 122
pre-login process, Liberty ID-FF, 161-164
principal, definition, 133-135
process
  *See* Federated Access Manager
  Liberty ID-FF, 161-164
provider federation, definition, 132-133
providers, 133-135

**R**

RADIUS authentication, 107
reader service, 160
realm authentication, policy, 121
realm-based authentication, 108
realm configuration, authentication, 110
realms, 51-54
  and access control, 122-123
redirection URLs, and authentication, 111
referral policy, 122
relational databases, logging, 192-193
remote logging, 195-196
resource, normal policy, 119
role-based authentication, 108
rule, policy, 119

**S**

SafeWord authentication, 107
SAML, convergence with Liberty ID-FF, 148-152
SAML authentication, 107
SAML v1.x
  architecture, 152-153
  federation, 146-154
  federation option, 145-146
  Liberty ID-FF comparison, 145-146
SAML v2, 148-152
  administration, 150-151
  API, 151
  basic configuration, 150-151
  features, 150
  federation, 146-154
  federation option, 145-146
  JSP, 152
  SPI, 151-152
secure logging, 195

SecurID authentication, 107
Security Token Service
  and Web Services Security, 47-48
  description, 47
service-based authentication, 109
service configuration plug-in, 64
Service Management Service, 64
service provider interface, *See* SPI
service provider interfaces, *See* SPI
service providers, definition, 133-135
services
  Authentication Service, 32-34
  Federated Access Manager, 31-50
  Federation Services, 44-46
  Identity Repository Service, 42-44
  Logging Service, 40-42
  Policy Service, 35-37
  Security Token Service, 47-48
  Session Service, 37-40
  Web Services Security, 47-48
session
  *See* user session
  basic user session, 83-93
  initial HTTP request, 83-85
session ID, *See* session token
session object, *See* session data structure
Session Service, description, 37-40
session service, overview, 79-80
session termination, 97-98
session token, 81-82
session tools, description, 63
session upgrade, and authentication, 105
session validation, process, 87-89
single sign-on, 155-156
  definition, 37-40, 80
  process, 93-95
single sign—on, and Liberty ID-FF, 161-164
SOAP Binding, 184
SOAP Binding Service, description, 47
SPI, 63-64
  Authentication Service, 64
  federation, 64
  SAML v2, 151-152
SSO, *See* single sign-on

subject, normal policy, 119-120

**T**
time, policy, 121
tools, 62-63
trust, definition, 133
trust agreements, 133

**U**
UNIX authentication, 107
user authentication, process, 85-87
user-based authentication, 109
user session
  cookies, 81-82
  definition, 80
  logging results, 91-93
  policy evaluation, 89-91
  session data structure, 81-82
  session termination, 97-98
  session token, 81-82
  session validation, 87-89
  user authentication, 85-87

**V**
value, normal policy, 119

**W**
web services
  implemented services, 184
  process, 184-186
Web Services Security, description, 47-48
web services security, Federated Access Manager, 22-23
web services stack, definition, 46-47
Windows Desktop SSO authentication, 108
Windows NT authentication, 108
writer service, 160

**X**
XACML, and authorization, 123-127
XML, files used, 46-47