Sun Java™ System

Access Manager 6 Developer's Guide

2005Q1

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

Part No: 817-7649

Contents

List of Figures

List of Tables

List of Procedures

List of Code Examples

Preface

The *Sun Java™ System Access Manager 6 2005Q1 Developer's Guide* offers information on how to customize Sun Java System Access Manager (formerly Sun™ ONE Identity Server) and integrate its functionality into an organization's current technical infrastructure. It also contains details about the programmatic aspects of the product and its APIs. Topics in this Preface include the following:

- "Who Should Use This Book" on page 27
- "Before You Read This Book" on page 28
- "Conventions Used in This Book" on page 28
- "Related Documentation" on page 31
- "Accessing Sun Resources Online" on page 33
- "Contacting Sun Technical Support" on page 33
- "Related Third-Party Web Site References" on page 34
- "Sun Welcomes Your Comments" on page 34

Who Should Use This Book

This *Developer's Guide* is intended for use by IT administrators and software developers who implement an integrated identity management and web access platform using Sun Java System servers and software. It is recommended that administrators understand the following technologies:

- Lightweight Directory Access Protocol (LDAP)
- Java™ technology
- JavaServer Pages™ (JSP) technology

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

Before You Read This Book

Access Manager is a component of Sun Java Enterprise System, a software infrastructure that supports enterprise applications distributed across a network or Internet environment. You should be familiar with the documentation provided with Sun Java Enterprise System, which can be accessed online at `http://docs.sun.com/coll/entsys_04q4`.

Because Sun Java System Directory Server is used as the data store in an Access Manager deployment, you should be familiar with the documentation provided with that product. Directory Server documentation can be accessed online at `http://docs.sun.com/coll/DirectoryServer_04q2`.

Conventions Used in This Book

The tables in this section describe the conventions used in this book.

Typographic Conventions

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

Table 1 Typographic Conventions

Typeface	Meaning	Examples
`AaBbCc123` (Monospace)	API and language elements, HTML tags, web site URLs, command names, file names, directory path names, onscreen computer output, sample code.	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`

`AaBbCc123` (Monospace bold)	What you type, when contrasted with onscreen computer output.	`%` **`su`**
		`Password:`

Table 1 Typographic Conventions *(Continued)*

Typeface	Meaning	Examples
AaBbCc123 (Italic)	Book titles, new terms, words to be emphasized. A placeholder in a command or path name to be replaced with a real name or value.	Read Chapter 6 in the *User's Guide*. These are called *class* options. Do *not* save the file. The file is located in the *install-dir*/`bin` directory.

Symbols

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

Table 2 Symbol Conventions

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

Default Paths and File Names

The following table describes the default paths and file names used in this book.

Example

Table 3 Default Paths and File Names

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

Shell Prompts

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

Table 4 Shell Prompts

Shell	Prompt
C shell on UNIX or Linux	*machine-name*%
C shell superuser on UNIX or Linux	*machine-name*#
Bourne shell and Korn shell on UNIX or Linux	`$`
Bourne shell and Korn shell superuser on UNIX or Linux	#
Windows command line	`C:\`

Related Documentation

To access Sun technical documentation online, go to http://docs.sun.com.

You can browse the documentation archive or search for a specific book title, part number, or subject.

Books in This Documentation Set

Table 5 Access Manager 6 2005Q1 Documentation Set

Title	Description
Technical Overview http://docs.sun.com/doc/817-7643	Provides a high-level overview of how Access Manager components work together to consolidate identity management and to protect enterprise assets and web-based applications. Explains basic Access Manager concepts and terminology
Deployment Planning Guide http://docs.sun.com/doc/817-7644	Provides information about planning a deployment within an existing information technology infrastructure
Administration Guide http://docs.sun.com/doc/817-7647	Describes how to use the Access Manager console as well as manage user and service data via the command line.
Migration Guide http://docs.sun.com/doc/817-7645	Describes how to migrate existing data and Sun Java System product deployments to the latest version of Access Manager. (For instructions about installing and upgrading Access Manager and other products, see the *Sun Java Enterprise System 2005Q1 Installation Guide*.)
Performance Tuning Guide http://docs.sun.com/doc/817-7646	Describes how to tune Access Manager and its related components.
Federation Management Guide http://docs.sun.com/doc/817-7648	Provides information about Federation Management, which is based on the Liberty Alliance Project.
Developer's Guide http://docs.sun.com/doc/817-7649	Offers information on how to customize Access Manager and integrate its functionality into an organization's current technical infrastructure. Contains details about the programmatic aspects of the product and its API.
Developer's Reference http://docs.sun.com/doc/817-7650	Provides summaries of data types, structures, and functions that make up the Access Manager public C APIs.

Table 5 Access Manager 6 2005Q1 Documentation Set *(Continued)*

Title	Description
Release Notes http://docs.sun.com/doc/817-7642	Available after the product is released. Contains 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.

Access Manager Policy Agent Documentation

Documentation for the Access Manager Policy Agents is available on the following documentation Web site:

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

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

- *Policy Agents For Web and Proxy Servers Guide* documents how to install and configure an Access Manager policy agent on various web and proxy servers. It also includes troubleshooting and information specific to each agent.
- *J2EE Policy Agents Guide* documents how to install and configure an Access Manager policy agent that can protect a variety of hosted J2EE applications. It also includes troubleshooting and information specific to each agent.
- The *Release Notes* are available online after a set of agents is released. The *Release Notes* include a description of what is new in the current release, known problems and limitations, installation notes, and how to report issues with the software or the documentation.

Other Server Documentation

For other server documentation, go to the following:

- Directory Server documentation
  http://docs.sun.com/coll/DirectoryServer_05q1
- Web Server documentation
  http://docs.sun.com/coll/WebServer_05q1
- Application Server documentation
  http://docs.sun.com/coll/ApplicationServer8_ee_04q4
- Web Proxy Server documentation
  http://docs.sun.com/prod/s1.webproxys#hic

Accessing Sun Resources Online

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

Download Center
http://wwws.sun.com/software/download/

Sun Java System Services Suite
http://www.sun.com/service/sunps/sunone/index.html

Sun Enterprise Services, Solaris Patches, and Support
http://sunsolve.sun.com/

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

Contacting Sun Technical Support

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

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

Related Third-Party Web Site References

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 document title and part number. The part number is a seven-digit or nine-digit number that can be found on the title page of the guide or at the top of the document.

For example, the title of this guide is *Sun Java System Access Manager 6 2005Q1 Developer's Guide*, and the part number is 817-7649.

Chapter 1

Introduction

The *Sun Java™ System Access Manager 6 2005Q1 Developer's Guide* describes the programmatic and customization details of Access Manager. It includes instructions on how to augment the application with new services using the eXtensible Markup Language (XML) files for configuration, the public Java™ application programming interfaces (APIs) for integration and the JavaServer Pages™ (JSP) for customization. This introductory chapter contains the following sections:

- "Access Manager Overview" on page 35
- "Extending Access Manager" on page 40
- "Access Manager File System" on page 43
- "Client Browser Support" on page 43

Access Manager Overview

Sun Java System Access Manager integrates identity management with the ability to create and enforce authentication processes and access to directory data and corporate resources. These capabilities enable organizations to deploy a comprehensive system that helps to secure and protect their assets and information, as well as deliver their web-based applications. Towards this end, Access Manager contains components and application management utilities or *services*.

NOTE An *identity* is a representation of an object used in a network environment. The identity, which can be internal (an employee, a printer) or external (a customer, a vendor), contains a set of attributes that uniquely identifies it. The simplest identity might contain user name (or object identifier) and password attributes. More complex identities might contain attributes for a phone number, social security number, building location, or address.

Data Management Components

Access Manager provides the following components to simplify the administration of identities and the management of data:

- Service Configuration—provides a solution for customizing and registering configuration parameters or *attributes* into a service; the service can then be integrated into, and managed using, Access Manager. The solution includes a Document Type Definition (DTD) that defines the structure for creating a service's XML file, Java APIs that are used to integrate the XML file into the deployment and the Access Manager console which is used to manage the service.
- Identity Management—provides a solution for managing identities. It includes an API for creating, modifying and removing Identity-related Objects (users, roles, groups, containers, organizations, sub-organizations, etc.) as well as an XML template that defines each object's Lightweight Directory Access Protocol (LDAP) attributes. This template allows for the object's storage in the Sun Java System Directory Server, the data store for Access Manager.
- Policy Management—provides a solution for defining and retrieving access privilege settings (or *policy*) to protect an enterprise's resources. It includes an API that applications can use to retrieve an identity's policy. The policy is then used to determine an identity's right to access the requested resource.
- Federation Management—provides a solution for defining authentication domains, service providers and identity providers in order to give users the functionality of *federation*. Federation allows a user to aggregate multiple digital identities allowing single sign-on to affiliated sites. This module is based on the Liberty Alliance Project's Version 1.1 specifications.
- Current Sessions—provides a solution for an Access Manager administrator to view and manage user session information. It keeps track of session times as well as allowing the administrator to terminate a session.

- Sun Java System Directory Server—provides the storage facility in an Access Manager deployment. It holds all identity data as well as configured policies. The majority of the data is stored in the Directory Server using LDAP; certain of it is stored as XML.

Access Manager Management Services

When Access Manager is installed, a number of utilities (or *services)* are installed to help manage the deployment. A *service* is actually a grouping of configuration parameters (or *attributes*). The attributes can be randomly grouped together for easy management or specifically grouped together for one purpose. Additional information on services can be found in Chapter 8, "Service Management," in this manual and the *Sun Java System Access Manager Administration Guide*. The current installed services include:

- Administration Service—provides properties for the configuration of the Access Manager as well as attributes to customize the application specific to each configured organization. Information on the Administration Service attributes can be found in the Administration Service attributes chapter of the *Sun Java System Access Manager Administration Guide*.
- Authentication Service—provides an interface for gathering user credentials and issuing single sign-on (session) tokens. It also contains an SDK to write plug-ins in order to integrate token validation and authentication credential storage functionality for proprietary authentication servers. For information on this service, see Chapter 5, "Authentication Service" of this manual and the chapter on the Authentication Service attributes in the *Sun Java System Access Manager Administration Guide*.
- Client Detection Service—allows Access Manager to detect the client type of an accessing browser. Information on this service can be found in Chapter 13, "Client Detection Service," in this manual and the chapter on the Client Detection Service attributes in the *Sun Java System Access Manager Administration Guide*.
- Globalization Settings—contains properties to configure Access Manager for different character sets. More information on this service, see the chapter on the Globalization Settings attributes in the *Sun Java System Access Manager Administration Guide*.

- Auditing Features—provides a record-keeping functionality. Both file-based logs and logs stored in a relational database are supported. Information on this service can be found in Chapter 12, "Auditing Features," in this manual and the chapter on the Logging Service attributes in the *Sun Java System Access Manager Administration Guide*.

- Naming Service—allows client browsers to locate the URL for services in a deployment that is running more than one Access Manager ensuring that the URL returned for the service is the one for the host on which the user session was created. More information on this service can be found in the Naming Service attributes chapter of the *Sun Java System Access Manager Administration Guide*.

- Password Reset Service—contains properties that can be configured per organization to implement the Password Reset Service. For information on this service, see the chapter on the Password Reset Service attributes in the *Sun Java System Access Manager Administration Guide*.

- Platform Service—provides configurable attributes for the Access Manager deployment. For information on this service, see the chapter on the Platform Service attributes in the *Sun Java System Access Manager Administration Guide*.

- Policy Configuration Service—provides properties for configuring the policy function as well as attributes to configure the Policy Service for each configured organization. For information on this service, see Chapter 9, "Policy Management," in this manual and the chapter on the Policy Configuration Service attributes in the *Sun Java System Access Manager Administration Guide*.

- Security Assertion Markup Language (SAML) Service—provides an interface integrating SAML service, Simple Object Access Protocol (SOAP) and `https` for sending and receiving security information. This service encrypts data passed between different security entities. An API is provided to this end. For information on this service, see Chapter 11, "SAML Service," in this manual and the chapter on the SAML Service attributes in the *Sun Java System Access Manager Administration Guide*.

- Session Service—provides attributes to configure session properties for all authorized sessions in each configured organization. For information on this service, see Chapter 4, "Single Sign-On And Sessions," in this manual and the chapter on the Session Service attributes in the *Sun Java System Access Manager Administration Guide*.

- User Service—provides attributes to configure the user properties for all users in each configured organization. For information on this service, see Chapter 7, "Identity Management," in this manual or the chapter on the User Service attributes in the *Sun Java System Access Manager Administration Guide*.

In addition to its configured services, Access Manager provides a graphical user interface that allows the application user to manage identity objects, services and policy information via a web browser. This console is built using the Sun Java System Application Framework and can be called by all users, from top level administrator to end users. The console can be customized for each configured organization by modifying and integrating a set of JSP and related files. Information on console customization can be found in Chapter 3, "The Access Manager Console," in this manual. Access Manager also offers data backup, restoration and other software utilities. Information on these functionalities can be found in Chapter 14, "Access Manager Utilities," in this manual. Information on command-line executables can be found in the *Sun Java System Access Manager Administration Guide*.

Managing Access

Access Manager can manage access to its protected resources in either of two ways: an user can authenticate and access Access Manager via a web browser or, an external application can access Access Manager directly, requesting user authentication information through the use of integrated Access Manager API.

Web Access

When a user requests access to a secure application or page using a web browser, they must first be authenticated. The request is directed to the Authentication Service which determines the type of authentication to initiate based on the method associated with the requestor's profile. For instance, if the user's profile is associated with LDAP authentication, the Authentication Service would send an HTML form to their web browser asking for an LDAP user name and password. (More complex types of authentication might include requesting information for multiple.) Having obtained the user's credentials, the Authentication Service calls the respective provider to verify the credentials. (The provider in the LDAP example would be the Directory Server.) Once verified, the service calls the SSO API to generate a Single Sign-On (SSO) or *session* token which holds the user's identity. The API also generates a *token ID*, a random identification string associated with the session token. The session token is then sent back to the requesting browser in the form of a cookie while the authentication component directs the user to the requested secure application or page. Additional information on the Authentication Service can be found in Chapter 5, "Authentication Service," in this manual.

NOTE Web access might also include an additional security measure to evaluate a user's access privileges. This includes installed policy agents. Additional information can be found in the *Sun Java System Access Manager Web Policy Agents Guide* and *J2EE Policy Agents Guide*.

Application Access

External applications can access Access Manager to request user information using the Access Manager SDK. For example, a mail service might store its users' mailbox size information in Access Manager and the SDK can be used to retrieve this information. To process the request, the system running the application must have the Access Manager SDK installed. Additional information on both the C and Java APIs can be found throughout this manual in the respective chapters.

Extending Access Manager

One of the architectural goals of Access Manager is to provide an extensible interface. This interface is defined by the following functions:

1. Custom services can be defined for the deployment using XML.
2. Console templates can be modified and/or customized for each organization using JSP.
3. Default services can be implemented using a set of Java API.

Service Definition With XML

Access Manager contains a number of management services. All Access Manager services are written using the XML. Administrators or service developers can modify the internal XML service files installed with Access Manager or configure new XML service files to customize the application based on their need. More information on services and how they are integrated into the Access Manager deployment can be found in Chapter 8, "Service Management," of this manual.

NOTE Access Manager services only *manage* attribute values that are stored in Sun Java System Directory Server. They do not implement their behavior or dynamically generate code to interpret them. It is up to an external application to interpret or utilize these values.

Console Customization

The Access Manager console is used for managing and monitoring identities, services and protected resources throughout the Access Manager deployment. The framework uses XML files, JSP templates and Cascading Style Sheets (CSS) to control the look and feel of the console screens. These files can be duplicated and then modified to make changes to the design for each configured organization; for instance, an organization’s logo can be added in place of the Sun logo. The entire template can also be replaced with an organization’s custom HTML page. Additional information on customizing the Access Manager console can be found in Chapter 3, “The Access Manager Console,” of this manual.

Access Manager SDK

The Access Manager SDK contains public interfaces to implement the behavior of Access Manager’s default or customized services. Both Java and C interfaces are provided. The packages include:

Identity Management SDK

Access Manager provides the framework to create and manage users, roles, groups, containers, organizations, organizational units, and sub-organizations. The Java package name is `com.iplanet.am.sdk`. There are currently no comparable C interfaces.

Service Management SDK

The service management interfaces can be used by developers to register services and applications, and manage their configuration data. The Java package name is `com.sun.identity.sm`. There are currently no comparable C interfaces.

Authentication Programming Interfaces

Access Manager provides interfaces to extend the functionality of the Authentication Service in two ways. The API provides interfaces that can be used remotely by either Java or C applications to utilize the authentication features of Access Manager. The SPI can be used to plug new authentication modules, written in Java, into the Access Manager authentication framework.

Utility API

This API provides a number of Java classes that can be used to manage system resources. It includes thread management and debug data formatting. The Java package name is `com.iplanet.am.util`. There are currently no comparable C interfaces.

Logging API And Logging SPI

The Logging Service records, among other things, access approvals, access denials and user activity. The Logging API can be used to enable logging for external Java applications. The package names begin with `com.sun.identity.log`. The Logging SPI are Java packages that can be used to develop plug-ins for customized features. The package names begin with `com.sun.identity.log.spi`. There are currently no comparable C interfaces.

Client Detection API

Access Manager can detect the type of client browser that is attempting to access its resources and respond with the appropriately formatted pages. The Java package used for this purpose is `com.iplanet.services.cdm`. There are currently no comparable C interfaces.

SSO API

Access Manager provides Java interfaces for validating and managing SSO tokens, and for maintaining the user's authentication credentials. All applications wishing to participate in the SSO solution can use this API. The Java package name is `com.iplanet.sso`. The Session Service also includes an API for C applications.

Policy SDK

The Policy API can be used to evaluate and manage Access Manager policies as well as provide additional functionality for the Policy Service. The Java package names begin with `com.sun.identity.policy`. The Policy Service also includes an API for C applications.

SAML SDK

Access Manager uses the SAML API to exchange acts of authentication, authorization decisions and attribute information. The Java package names begin with `com.sun.identity.saml`. There are currently no comparable C interfaces.

Federation Management API

Access Manager uses the Federation Management API to add functionality based on the Liberty Alliance Project specifications. The Java package name is `com.sun.liberty`. There are currently no comparable C interfaces.

Access Manager File System

Access Manager installs its packages and files in a directory named `SUNWam`. The complete file system layout for Access Manager can be found in the *Sun Java System Access Manager Deployment Guide*.

Client Browser Support

Access Manager 2005Q1 is supported on the following client browsers:

- Netscape™ Communciator 7.0
- Netscape Communicator 6.2.1
- Netscape Navigator™ 4.79
- Microsoft® Internet Explorer 6.0
- Microsoft Internet Explorer 5.5

Chapter 2

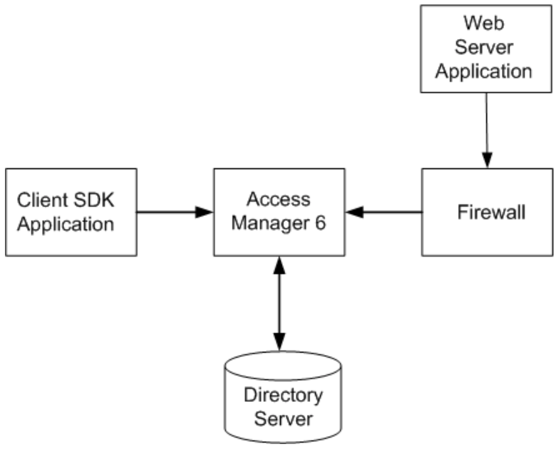
Using the Client SDK

The Sun Java™ System Access Manager 6 2005Q1 Client SDK package provides Access Management Java libraries for implementing stand-alone and web applications. You can use the Client SDK interfaces in your applications to take advantage of Access Manger services such as authentication, Single Sign-On (SSO), authorization, auditing and logging, user management, and Security Assertion Markup Language (SAML). The client SDK libraries communicate with Access Manager using XML (SOAP) over HTTP or HTTPS.

How the Client SDK Works

The Client SDK is different from the SDK packages provided in previous versions of Access Manager. The Access Manager 6.3 Client SDK has been streamlined to include only the client-side classes and configuration properties you need to access Access Manager services. These changes result in a smaller jar file, and eliminate the dependency on connections to Directory Server when developing and deploying client applications. In the Access Manager 6.3 architecture, the Client SDK and client applications communicate with the Access Manager server. Only the Access Manager server communicates directly with the Directory Server.

Figure 2-1 Client SDK Architecture



JDK and CLASSPATH Requirements

The Client SDK can be used with JDK versions J2SE 1.3.2, J2SE 1.4.2 and higher.

To use the Client SDK with JDK 1.3.2 add the following to the `CLASSPATH`:

Java Authentication and Authorization Service (JAAS). Available at the following URL: `JAAS hhtp://java.sun.com/products/jaas/`

Java Web Services Developer Pack 1.3 (Java WSDP). Available at the following URL: `Java WSDP http://java.sun.com/products/jwsdp/`

Java Secure Socket Extension (JSSE). Available at the following URL: `http://java.sun.com/products/jsse/`

JDK Logging. The jar `jdk_logging.jar` can be obtained from `SUNWamsdk` package for Solaris and `sun-identity-sdk` RPM for Linux

amclientsdk.jar. This jar is located in the directory *AccessManager-base*`/SUNWam/lib`.

servlet.jar. This jar can be obtained as part of the SUNWamsdk package, or from the *AccessManager-base*/SUNWam/lib directory if Access Manger is installed.

NOTE To use the Client SDK with JDK 1.4.2 and higher versions, only amclientsdk.jar and servlet.jar are required in the CLASSPATH.

Configuring the Client SDK

Before installing the Client SDK, an instance of Access Manager must be running, and you must know the URL for accessing it. The client SDK libraries use this URL to communicate with Access Manager using XML (SOAP) over HTTP or HTTPS. The Client SDK is contained in the following file:

AccessManager-base/SUNWam/lib/amclientsdk.jar

Table 2-1 summarizes items included in the Client SDK.

Table 2-1 Contents of *AccessManager-base*/SUNWam/amclientsdk.jar

File	Description
README.clientsdk	ASCII version of this chapter. Contains information on installing and using Access Manager client SDK.
lib/amclientsdk.jar	Client SDK for stand-alone applications.
amclient.war	Archive of Access Manager samples, web applications, and Javadoc.
Makefile.clientsdk	Defines objects and parameters for building sample properties, stand-alone samples and web applications.

To Configure the Client SDK

1. In Makefile.clientsdk, edit the following parameters to suit your environment:

   JAVA_HOME

   SERVER_HOSTNAME

   SERVER_PORT

2. If implementing User Management, SAML, or Policy, then edit the following parameters to suit your environment:

APPLICATION_USERNAME

APPLICATION_PASSWORD

If an encrypted password or secret exists, then provide the following instead of `ADMIN_PASSWORD`:

ENCRYPTED_PASSWORD

ENCRYPTION_KEY

3. Run the make command:

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

This generates a sample properties file in the directory `temp`, standalone samples in the directory `clientsdk-samples` and a deployable war file, `amclientwebapps.war`, that can be deployed in any Servlet 2.3 compliant container. Table 2-2 summarizes the items included in the war file.

Table 2-2 Contents of `AccessManager-base/SUNWam/amclientsdk.war`

File	Description
`index.html`	Instructions for installing and using the Client SDK packages
`WEB-INF/web.xml`	Client SDK for stand-alone applications
`WEB-INF/classes/AMClient.properties`	Archive of Access Manager samples, web applications, and Javadoc
`WEB-INF/classes/*.classes`	File for building stand-alone samples and web applications
`WEB-INF/docs`	Javadoc (Public Client SDK APIs)
`WEB-INF/samples`	Sample stand-alone programs
`WEB-INF/webapps`	Sample web applications

Initializing the Client SDK

Before Access Manager Client SDK can communicate with Access Manager Server, you must initialize some properties in the client SDK. You can set these properties in one of three ways:

- Using a Properties File
- Using the Java API
- Setting Individual Properties

Using a Properties File

You can set properties in a properties file and then provide a path to it at runtime. The properties files must be in the CLASSPATH. The default properties file name is `AMConfig.properties` and is always read at start-up.

To Set ClientSDK Properties in a Properties File

1. Generate a sample `AMConfig.properties` by running the following command:

   ```
   make -f Makefile.clientsdk properties
   ```

   The `AMConfig.properties` will be present in the `temp` directory.

2. Edit properties to suit your environment.

3. At runtime if the file name is different from `AMConfig`, provide the edited properties filename (without the `.properties` extension, and also with the path. The path should be in the CLASSPATH) by declaring the JVM option:

   `-Damconfig=`*filname*

Using the Java API

The ClientSDK properties can also be set programatically using the class: `com.iplanet.am.util.SystemProperties`. See Code Example 2-1:

Code Example 2-1 Setting ClientSDK Properties

```
import com.iplanet.am.util.SystemProperties;
import java.util.Properties;
public static void main(String[] args) {
    // To initialize a set of properties
    Properties props = new Properties();
    props.setProperty(‘com.iplanet.am.naming.url’,
‘http://sample.com/amserver/namingservice’);
    props.setProperty(‘com.sun.identity.agents.app.username’, ‘amAdmin’);
    props.setProperty(‘com.iplanet.am.service.password’, ‘11111111’);
    SystemProperties.initializeProperties
(props) ;

    // To initialize a single property
    SystemProperties.initializeProperties(“com.iplanet.am.naming.url’,
‘http://sample.com/amserver/namingservice’);
    // Application specific code ...
}
```

Setting Individual Properties

You can set properties one at a time. For example, you can declare the following JVM option at run time to assign a value to a particular property:

-D*propertyName*=*propertyValue*

The following sections describe the properties expected by Access Manager Client SDK. A client application deployed within a servlet container can register for changes to session, user attributes and policy decisions.These properties must be set to receive such notifications.

Naming URL Properties

com.iplanet.am.naming.url. This is a required property. The value of this property represents the URL where the Client SDK would retrieve the URLs of Access Manager internal services. This is the URI for the Naming Service. Example:

`com.iplanet.am.naming.url=http://`*access_manager_host.domain_name*`:`*port*`/`
`amserver/namingservice`

com.iplanet.am.naming.failover.url. This is a required property. This property can be used by any remote SDK application that wants failover in, for example, session validation or getting the service URLs. Example:

```
com.iplanet.am.naming.failover.url=
    http://access_manager_host.domain_name:port/amserver/failover
```

Debug Properties

com.iplanet.services.debug.level. Specifies the debug level.Possibe values are levels are: `off`, `error`, `warning`, or `message`.

com.iplanet.services.debug.directory. The value of this property is the output directory for the debug information. This directory should be writable by the server process. Example:

```
com.iplanet.services.debug.directory=/var/opt/SUNWam/debug
```

Notification URL Properties

com.iplanet.am.notification.url.

The value of this property is the URI of the Notification Service running on the host machine where you installed the Client SDK. Example:

```
com.iplanet.am.notification.url=
 http://clientSDK_host.domain_name:port/amserver/notificationservice
```

com.sun.identity.agents.notification.enabled. This property enable or disables notifications for remote policy API. Example:

```
com.sun.identity.agents.notification.enabled=false
```

com.sun.identity.agents.notification.url. This property defines the notification URL for remote policy API.

Setting Up a Client Identity

Some of the Access Manager components such as SAML, User Management, Policy, require an identity for the client. The client application reads configuration data to identify the client. You can set up the identity for the client in one of two ways:

- Set username and password properties can be authenticated
- Set an SSO Token Provider

NOTE Some of the configuration attributes (such as password) are encrypted and stored in the data store as an Encryption/Decryption Key. If such attributes have to be decrypted by the client, the property must be set, and must be the same as that of the Access Manager Server.

This value is generated at installation time and stored in `/IdentityServer_base/SUNWam/lib/AMConfig.properties`. More information on this property can be found in the "Encryption" section of the Appendix A, "AMConfig.properties File."

To Set Username and Password Properties

The following properties can be used to set the username and password that can be used by client SDK to obtain the configuration parameters. The authenticated username should have permissions to read the configuration data for SAML and User Management.

- The property to provide the user name is:
  `com.sun.identity.agents.app.username`
- The property to provide the plain text password is:
  `com.iplanet.am.service.password`

For scenarios where plain text password would be security concern, an encrypted password can be provided using the property: `com.iplanet.am.service.secret`

If an encrypted password is provided, the encryption key must also be provided using the property: `am.encryption.pwd`

To Set an SSO Token Provider

Set the following property: `com.sun.identity.security.AdminToken`

This provides an implementation for the the interface, which returns the following single-sign-on (SSO) token: `com.sun.identity.security.AppSSOTokenProvider`.

Building Custom Web Applications

The Client SDK package contains `Makefile.clientsdk` that you can use to generate and build samples and web applications. The makefile defines targets to build configuration properties, samples and web applications.

Building Stand-Alone Applications

Use these steps a template for building their identity-enabled web applications.

To Build a Stand-Alone Application

1. Install the Client SDK.

   Follow the steps in the section“To Configure the Client SDK” on page 47.

2. Copy `servlet.jar` to ../lib directory.

3. If using JDK 1.3, follow these steps:

   a. copy the following jars to the ../lib directory:

      - `jaas.jar`
      - `jsse.jar jce1_2_1.jar`
      - `jdk_logging.jar`

   b. Add the jar files the CLASSPATH definition in the file `clientsdk-samples/defines.mk.`

4. Run the stand-alone application.

   Change directory to respective components within `clientsdk-samples`. Each has a `Readme.html` file explaining the changes to done and a Makefile to rebuild and run the program.

Targets Defined in clientsdk

For web deployment, `amclientwebapps.war` is ready to be deployed. However, you can make changes in `clientsdk-webapps` directory and the war file can be recreated.

Custom web applications can use the following as a template to build their identity enabled web application.

properties: Generates `AMConfig.properties` in the temp directory that can used as a template for setting AM SDK's properties

samples: Copies standalone samples and corresponding Makefiles to samples directory.

webapp: Generates `amclientwebapps.war` that can be deployed on any Servlet 2.3 compliant web container.

About the Client SDK Samples

Sample files are included in the Client SDK. These demonstrate how to write stand-alone programs and how to write web applications.The samples are located under the directory where you generated the `Makefile.clientsdk`, and in the following subdirectories:

```
.../clientsdk-samples/
```

```
.../clientsdk-webapps/
```

`Clientsdk-samples` includes samples for authentication, logging, policy and SAML stand-alone programs.`Clientsdk-webapps` includes samples `for` user management, service management, and policy programs. Each sample has a `Readme.html` file with instructions on compiling and running the sample program..

Chapter 3

The Access Manager Console

The Sun Java™ System Access Manager 6 2005Q1 console is a web-based interface for creating, managing, and monitoring the identities, web services, and enforcement policies configured throughout an Access Manager deployment. It is built with Sun Java System Application Framework, a Java™ 2 Enterprise Edition (J2EE) framework used to help developers build functional web applications. XML files, JavaServer Pages™ (JSP) and Cascading Style Sheets (CSS) are used to define the look of the HTML pages. This chapter explains the console, its pluggable architecture, and how to customize it. It contains the following sections:

Overview

The Access Manager console is a web interface that allows administrators with different levels of access to, among other things, create organizations, create (and delete) users to (and from) those organizations, and establish enforcement policies that protect and limit access to the organization's resources. In addition, administrators can view and terminate current user sessions and manage their federation configurations (create, delete and modify authentication domains and providers). Users without administrative privileges, on the other hand, can manage personal information (name, e-mail address, telephone number, etc.), change their password, subscribe and unsubscribe to groups, and view their roles. All of these functionalities are accomplished using a web browser.

NOTE The client web browser accessing the console must support JavaScript, version 1.2 and cookies.

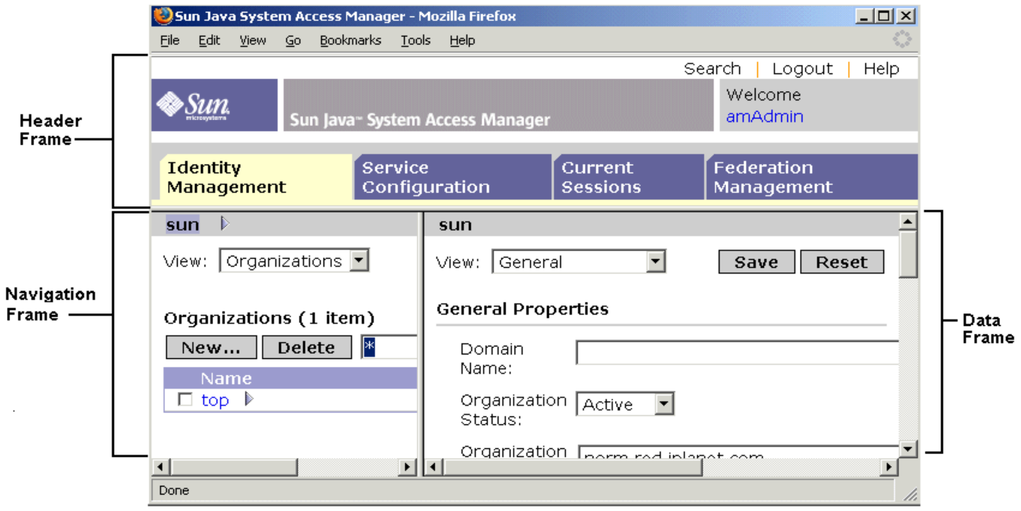
The console ships with four modules: Identity Management (including user and policy management), Service Configuration, Current Sessions (including session management) and Federation Management. Customization of these modules and the Access Manager console can be achieved, in varying degrees, by modifying the JSP and XML files that define the interface as well as extending the Sun Java System Application Framework ViewBeans.

NOTE A ViewBean is a Java class written specifically for rendering display. In Access Manager, each identity object has its own profile ViewBean. For example, the user profile has the `UMUserProfileViewBean`.

Console Interface

The console is divided into three frames as pictured in Figure 3-1: Header, Navigation and Data. The Header frame displays corporate branding information as well as the first and last name of the currently logged-in user as defined in their profile. It also contains a set of tabs to allow the user to switch between the management modules, a hyperlink to the Access Manager Help system, a Search function and a Logout link. The Navigation frame on the left displays the object hierarchy of the chosen management module, and the Data frame on the right displays the attributes of the object selected in the Navigation frame.

Figure 3-1 The Access Manager Console



Generating The Console Interface

When the Access Manager console receives an HTTP(S) request, it first determines whether the requesting user has been authenticated. If not, the user is redirected to the Access Manager login page supplied by the Authentication Service. After successful authentication, the user is redirected back to the console which reads all of the user's available roles, and extracts the applicable permissions and behaviors. The console is then dynamically constructed for the user based on this information. For example, users with one or more administrative roles will see the administration console view while those without any administrative roles will see the end user console view. Roles also control the actions a user can perform and the identity objects that a user sees. Pertaining to the former, the organization administrator role allows the user read and write access to all objects within that organization while a help desk administrator role only permits write access to the users' passwords. With regards to the latter, a person with a people container administrator role will only see users in the relevant people container while the organization administrator will see all identity objects. Roles also control read and write permissions for service attributes as well as the services the user can access.

Plug-In Modules

An external application can be plugged-in to the console as a module, gaining complete control of the Navigation and Data frames for its specific functionality. In this case, a tab with the name of the custom application needs to be added to the Header frame. The application developer would create the JSPs for both left and right frames, and all view beans, and models associated with them. Information on how to define a module tab can be found in "To Add A Module Tab" on page 68.

Accessing The Console

The Naming Service defines URLs used to access the internal services of Access Manager. The URL used to access the Administration Console web application is:

`http://`*identity_server_host.domain_name:port*`/amconsole`

The first time Administration Console (`amconsole`) is accessed, it brings the user to the Authentication web application (`amserver`) for authentication and authorization purposes. After login, `amserver` redirects the user to the configured success login URL as discussed in "The User Interface Login URL" on page 77 of Chapter 5, "Authentication Service." The default successful login URL is `http(s)://`*identity_server_host.domain_name:port*`/amconsole/base/AMAdminFrame`.

Customizing The Console

The Access Manager console uses JSP and CSS to define the look and feel of the pages used to generate its frames. A majority of the content is generated dynamically—based on where, and at what, the user is looking. In that regard, the modification of the content is somewhat restricted. Within the Navigation frame, the layout of the controls (the view menu), the action buttons, and the table with current objects in each JSP can be changed. In the Data frame, the content displayed is dynamically generated based on the XML service file being accessed but the layout, colors, and fonts are controlled by the `adminstyle.css` style sheet.

The Default Console Files

An administrator can modify the console by changing tags in the JSP and CSS. All of these files can be found in the *IdentityServer_base*/`SUNWam/web-src/applications/console` directory. The files in this directory provide the default Sun Java System interface. Out of the box, it contains the following sub-directories:

- `base` contains JSP that are not service-specific.
- `css` contains the `adminstyle.css` which defines styles for the console.
- `federation` contains JSP related to the Federation Management module.
- `html` contains miscellaneous HTML files.
- `images` contains images referenced by the JSP.
- `js` contains JavaScript™ files.
- `policy` contains JSP related to the Policy Service.
- `service` contains JSP related to the Service Management module.
- `session` contains JSP related to the Current Sessions (session management) module.
- `user` contains JSP related to the Identity Management module.

NOTE Console-related JSP contain HTML and custom library tags. The tags are defined in tag library descriptor files (`.tld`) found in the *IdentityServer_base*/`SUNWam/web-src/WEB-INF` directory. Each custom tag corresponds to a view component in its view bean. While the tags in the JSP can be removed, new tags can not be added. For more information, see the Sun Java System Application Framework documentation.

Creating Custom Organization Files

To customize the console for use by a specific organization, the *IdentityServer_base*/`SUNWam/web-src/applications/console` directory should first be copied, renamed and placed on the same level as the default directory. The files in this new directory can then be modified as needed.

NOTE There is no standard to follow when naming the new directory. The new name can be any arbitrarily chosen value.

For example, customized console files for the organization `dc=new_org, dc=com` might be found in the *IdentityServer_base*`/SUNWam/web-src/applications/custom_directory` directory.

To Create Custom Organization Files

1. Change to the directory where the default templates are stored:

`cd` *IdentityServer_base*`/SUNWam/web-src/applications`

2. Make a new directory at that level.

The directory name can be any arbitrary value. For this example, it is named *IdentityServer_base*`/SUNWam/web-src/applications/custom_directory/`.

3. Copy all the JSP files from the `console` directory into the new directory.

IdentityServer_base`/SUNWam/web-src/applications/console` contains the default JSP for Access Manager. Ensure that any image files are also copied into the new directory.

4. Customize the files in the new directory.

Modify any of the files in the new directory to reflect the needs of the specific organization.

5. Modify the `AMBase.jsp` file.

In our example, this file is found in *IdentityServer_base*`/SUNWam/web-src/applications/custom_directory/bas` E. The line `String console = "../console";` needs to be changed to `String console = "../`*new_directory_name*`";`. The `String consoleImages` tag also needs to be changed to reflect a new image directory, if applicable. The contents of this file are copied in Code Example 3-1.

Code Example 3-1 The AMBase.jsp File

```
<!--
 Copyright © 2002 Sun Microsystems, Inc. All rights reserved.
 Use is subject to license terms.
-->

<% String console = "../console";
   String consoleUrl = console + "/";
   String consoleImages = consoleUrl + "images";
%>
```

6. Change the value of the JSP Directory Name attribute in the Administration Service to match that of the directory created in Step 2 on page 60.

   The JSP Directory Name attribute points the Authentication Service to the directory which contains an organization's customized console interface. Using the console itself, display the services registered to the organization for which the console changes will be displayed. If the Administration Service is not visible, it will need to be registered. For information on registering services, see Chapter 8, "Service Management" in this manual or the *Sun Java System Access Manager Administration Guide*.

Once the new set of console files have been modified, the user would need to log into the organization where they were made in order to see any changes. Elaborating on our example, if changes are made to the JSP located in the *IdentityServer_base*/`SUNWam/web-src/applications/custom_directory` directory, the user would need to login to that organization using the URL `http://`*server_name.domain_name:port*`/`*service_deploy_uri*`/UI/Login?org=`*custom_directory_organization*.

NOTE More information on this login URL and authentication URL parameters can be found in Chapter 5, "Authentication Service" in this manual.

Alternate Customization Procedure

The console can also be modified by simply replacing the default images in *IdentityServer_base*/`SUNWam/web-src/applications/console/images`, with new, similarly named images.

Miscellaneous Customizations

Included in this section are procedures for several specific customizations available to administrators of the Access Manager console.

To Modify The Service Configuration Display

A *service* is a group of attributes that are managed together by the Access Manager console. Out-of-the-box, Access Manager loads a number of services it uses to manage its own features. For example, the configuration parameters of the Logging Service are displayed and managed in the Access Manager console, while code implementations within Access Manager use the attribute values to run the service. There is a defined procedure for adding Access Manager services to the console. For information on this procedure, see "Defining A Custom Service" on page 187 of Chapter 8, "Service Management." Chapter 8 also contains information on how to extend existing services, add or remove a service name from the Navigation frame using the "serviceHierarchy Attribute" and change the default service display using the "propertiesViewBeanURL Attribute"

To Modify The User Profile View

The Access Manager console creates a default User Service view based on information defined in the `amUser.xml` service file.

NOTE Attributes defined as User attributes in each service's specific XML file can also be displayed in the User Service. More information on how this is done can be found in "Customizing User Pages" on page 237 of Chapter 8, "Service Management" in this manual.

A modified user profile view with functionality more appropriate to the organization's environment can be defined by creating a new ViewBean and/or a new JSP. For example, an organization might want User attributes to be formatted differently than the default vertical listing provided. Another customization option might be to break up complex attributes into smaller ones. Currently, the server names are listed in one text field as:

protocol://*Access Manager_host*.*domain*:*port*

Instead, the display can be customized with three text fields:

protocol_chooser_field://*server_host_field*:*port_number_field*

A third customization option might be to add JavaScript to the ViewBean to dynamically update attribute values based on other defined input. The custom JSP would be placed in the following directory: *IdentityServer_base*`/SUNWam/web-src/applications/console/user.` The ViewBean is placed in the classpath `com.iplanet.am.console.user`. The value of the attribute User Profile Display Class in the Administration Service (`iplanet-am-admin-console-user- profile-class` in the `amAdminConsole.xml` service file) would then be changed to the name of the newly created ViewBean. The default value of this attribute is `com.iplanet.am.console.user.UMUserProfileViewBean`. More information on this procedure can be found in "Console Samples" on page 70.

Display Options For The User Profile Page

There are a number of attributes in the Administration Service that can be selected to display certain objects on the User Profile page. Display User's Roles, Display User's Groups and User Profile Display Options specify whether to display the roles assigned to a user, the groups to which a user is a member and the schema attributes, respectively. More information on these service attributes can be found in the *Sun Java System Access Manager Administration Guide*.

To Localize The Console

All textual resource strings used in the console interface can be found in the `amAdminModuleMsgs.properties` file, located in *IdentityServer_base*`/SUNWam/locale/`. The default language is English (`en_US`). Modifying this file with messages in a foreign language will localize the console.

To Display Service Attributes

Service attributes are defined in XML service files based on the `sms.dtd`. In order for a particular service attribute to be displayed in the console, it must be configured with the `any` XML attribute. The any attribute specifies whether the service attribute for which it is defined will display in the Access Manager console. More information on this attribute can be found in "any Attribute" on page 208 of Chapter 8, "Service Management" in this manual.

To Customize Interface Colors

All the colors of the console are configurable using the Access Manager style sheet `adminstyle.css` located in the *IdentityServer_base*`/SUNWam/web-src/applications/console/css` directory. For instance, to change the background color for the navigation frame, modify the `BODY.navFrame` tag; or to change the background color for the data frame, modify the `BODY.dataFrame`. The tags take either a text value for standard colors (blue, green, red, yellow, etc.) or a hexadecimal value (#ff0000, #aadd22, etc.). Replacing the default with another value will change the background color of the respective frame after the page is reloaded in the browser. Code Example 3-2 details the tag in `adminstyle.css`.

Code Example 3-2 `BODY.navFrame` Portion of `adminstyle.css`

```
BODY.navFrame   {
                color: black;
                background: #ffffff;
                }
```

To Change The Default Attribute Display Elements

The console auto-generates Data frame pages based on the definition of a service's attributes in an XML service definition file. As documented in "The sms.dtd Structure" in Chapter 8, "Service Management" in this manual, each service attribute is defined with the XML attributes `type`, `uitype` and `syntax`. `Type` specifies the kind of value the attribute will take. `uitype` specifies the HTML element displayed by the console. `syntax` defines the format of the value. The values of these attributes can be mixed and matched to alter the HTML element used by the console to display the values of the attributes. For example, by default, an attribute of the `single_choice` type displays its choices as a drop down list in which only one choice can be selected. This list can also be presented as a set of radio buttons if the value of the `uitype` attribute is changed to `radio`. Code Example 3-3 illustrates this concept.

Code Example 3-3 `uitype` XML Attribute Sample

```
<AttributeSchema name="test-attribute"
                   type="single_choice"
                   syntax="string"
                   any="display"
                   uitype="radio"
                   i18nKey="d105">
                   <ChoiceValues>
<ChoiceValue i18nKey="u200">Daily</ChoiceValue>
<ChoiceValue i18nKey="u201">Weekly</ChoiceValue>
<ChoiceValue i18nKey="u202">Monthly</ChoiceValue>
                   </ChoiceValues>
                   <DefaultValues>
                       <Value>Daily</Value>
                   </DefaultValues>
               </AttributeSchema>
```

Table 3-1 is a listing of the possible values for each attribute, and the corresponding HTML element that each will display based on the different groupings.

Table 3-1 Service Attribute Values and Corresponding Display Elements

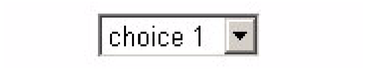
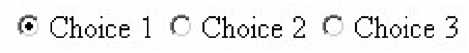
type Value	syntax Value	uitype Value	Element Displayed In Console
single_choice	string	No value defined	pull-down menu choices 
		radio	radio button choices 

Table 3-1 Service Attribute Values and Corresponding Display Elements *(Continued)*

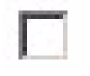
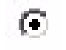
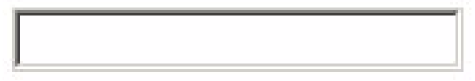
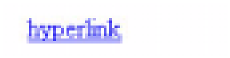
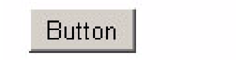
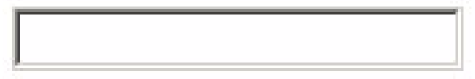
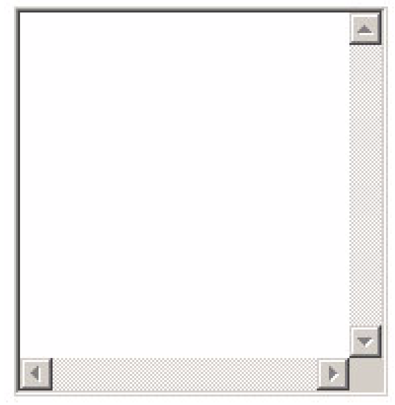
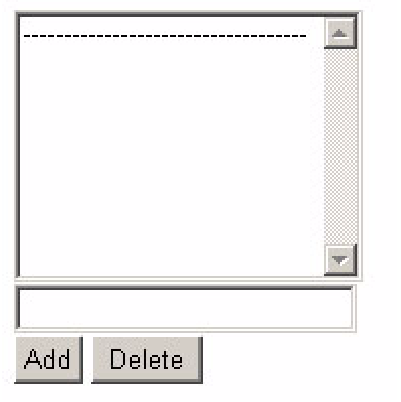
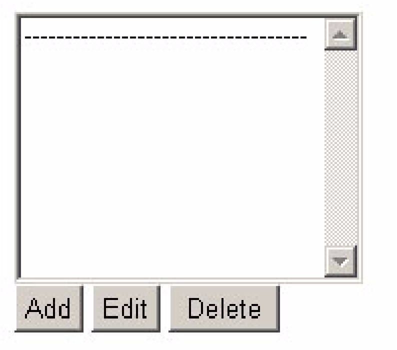
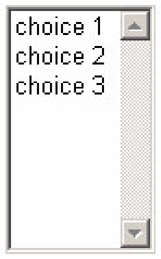
type Value	syntax Value	uitype Value	Element Displayed In Console
Single	boolean	No value defined	checkbox 
		radio	radio button 
	string	No value defined	text field 
		link	hyperlink 
		button	clickable button 
	password	No value defined	text field 
	paragraph	No value defined	scrolling text field 

Table 3-1 Service Attribute Values and Corresponding Display Elements *(Continued)*

type Value	syntax Value	uitype Value	Element Displayed In Console
list	string	No value defined	Add/Delete name list 
		name_value_list	Add/Edit/Delete name list 
multiple_choice	string	No value defined	choice list 

To Add A Module Tab

"Plug-In Modules" on page 58 mentions the capability to plug-in external applications as modules. Once this is accomplished, the module needs to be accessible via the console by adding a new module tab. Label information for module tabs are found in the `amAdminModuleMsgs.properties` console properties file located in *IdentityServer_base*`/SUNWam/locale/`. To add label information for a new module, add a key and value pair similar to `module105_NewTab=My New Tab`. Code Example 3-4 illustrates the default pairs in the file.

Code Example 3-4 Module Tab Key And Value Pairs

```
module101_identity=Identity Management
module102_service=Service Configuration
module103_session=Current Sessions
module104_federation=Federation Management
```

The module name and a URL for the external application also need to be added to the View Menu Entries attribute in the Administration Service (or `iplanet-am-admin-console-view-menu` in the `amAdminConsole.xml` service file). When a module tab in the Header frame is clicked, this defined URL is displayed in the Navigation frame. For example, to define the display information for the tab sample, an entry similar to `module105_NewTab|/amconsole/`*custom_directory*`/`*custom_NavPage* would be added to the View Menu Entries attribute in the Administration Service.

NOTE The console retrieves all the entries from this attribute and sorts them by i18n key. This determines the tab display order in the Header frame.

After making these changes and restarting Access Manager, a new tab will be displayed with the name My New Tab. For information on the sample that explains how to add a new tab, see "Console Samples" on page 70.

To Display Container Objects

In order to create and manage LDAP organizational units (referred to as *containers* in the console), the following attributes need to be enabled (separately or together) in the Administration Service.

- Display Containers In Menu—Containers are organizational units as viewed using the Access Manager console. If this option is selected, the menu choice Containers will be displayed in the View menu for top-level Organizations, Sub-Organizations and other containers.
- Show People Containers—People containers are organizational units containing user profiles. If this option is selected, the menu choice People Containers will be displayed in the View menu for Organizations, Containers and Sub-Organizations.
- Show Group Containers—Group containers are organizational units containing groups. If this option is selected, the menu choice Group Containers will be displayed in the View menu for Organizations, Containers and Group Containers.

Viewing any of these display options is also dependent on whether the Enable User Management attribute is selected in the Administration Service. (This attribute is enabled by default after a new installation.) More information on these attributes can be found in the *Sun Java System Access Manager Administration Guide*.

Console API

The public console API package is named `com.iplanet.am.console.base.model`. It contains interfaces that can be used to monitor and react to events that occur in the console. This *listener* can be called when the user executes an action on the console that causes an event. An event can have multiple listeners registered on it. Conversely, a listener can register with multiple events. Events that might be used to trigger a listener include:

- Displaying a tab in the Header frame.
- Creating or deleting identity-related objects.
- Modifying the properties of an identity-related object.
- Sending attribute values to the console ViewBean for display purposes.

When a listener is created all the methods of that interface must be implemented thus, the methods in the `AMConsoleListener` interface must be implemented. The `AMConsoleListenerAdapter` class provides default implementations of those methods and can be used instead. Creating a console event listener includes the following:

1. Write a console event listener class (or implement the default methods in the `AMConsoleListenerAdapter` class).

2. Compile the code.
3. Register the listener in the Administration Service.

Access Manager includes a sample implementation of the ConsoleEventListener. See "ConsoleEventListener" on page 71 for more information. The Access Manager Javadocs also contains more detailed information on the listener interfaces and class.

Precompiling The Console JSP

Each JSP is compiled when it is first accessed. Because of this, there is a delay when displaying the HTML page on the browser. To avoid this delay, the system administrator can precompile the JSP by running the following command:

WebServer_install_directory`/servers/bin/https/bin/jspc -webapp` *IdentityServer_base*`/SUNWam/web-src/applications`

where, by default, *WebServer_install_directory* is `/opt/SUNWwbsvr`.

Console Samples

Sample files have been included to help understand how the Access Manager console can be customized. The samples include instructions on how to:

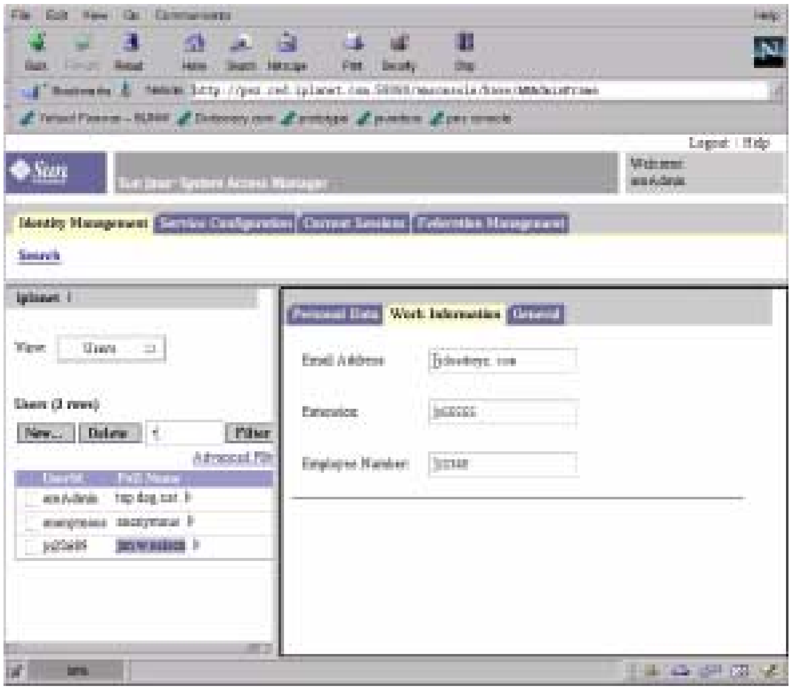
Modify User Profile Page

This sample modifies the user interface by adding a hyperlink that allows an existing user to change their configured password. It is in the `ChangeUserPassword` directory.

Create A Tabbed Identity Management Display

This sample creates a custom user profile which displays the profile with three tabs. Figure 3-2 contains a screenshot of a tabbed user profile. It is in the `UserProfile` directory.

Figure 3-2 Console With Three Tabs



ConsoleEventListener

This sample displays the parameters passed to `AMConsoleListener` class in the `amConsole` debug file. It is in the `ConsoleEventListener` directory.

Add Administrative Function

This sample adds functionality to the Identity Management module that allows an administrator to move a user from one organization to other. It is in the `MoveUser` directory.

Add A New Module Tab

This sample adds a new tab into the Header frame. This tab will connect to an external application and can be configured using the console. It is in the `NewTab` directory.

Create A Custom User Profile View

This sample creates a custom user profile view to replace the default user profile view. A different user profile view can be created for each configured organization. A custom class would need to be written that extends the default user profile view bean. This class would then be registered in the User Profile Display Class attribute of the Administration Service. There is an example of how to do this in the samples directory. This sample is in the `UserProfile` directory.

These samples are located in *IdentityServer_base*`/SUNWam/samples/console`. Open the `README` file in this directory for general instructions. Each specific sample directory also contains a `README` file with instructions relevant to that sample.

NOTE The console samples are only available when Access Manager is installed on the Solaris™ operating system.

Chapter 4

Single Sign-On And Sessions

The Session Service is a key component of the Sun Java™ System Access Manager 6 2005Q1 single sign-on (SSO) solution that enables users to authenticate once yet access multiple resources. In other words, successive attempts by a user to access protected resources will not require them to provide authentication credentials for each attempt. This chapter explains the Session Service, the SSO solution, and the SSO APIs. It contains the following sections:

Overview

A user wanting to access resources protected by Access Manager must first pass validating credentials through the Authentication Service. A successful authentication gives the user authorization to access the protected resources, based on their assigned access privileges or *policy*. If a user wants to access *several* resources protected by Access Manager, the Session Service provides proof of authorization so there is no need to re-authenticate; this is *single sign-on*. As different DNS domains generally have common users who need to gain access to their services in a single session, Access Manager supports a *cross-domain single sign-on* functionality.

NOTE In an Access Manager deployment, all Access Manager instances must be located in one primary cookie domain. The deployment may have multiple instances for high-availability but they may not be located in multiple DNS domains.

The Session Service provides the functionality to maintain information about an authenticated user's session across all applications participating in a single sign-on. It is responsible for:

- Generating session identifiers.
- Maintaining a master copy of the session's state information.
- Implementing the time-dependent behavior of sessions.
- Implementing the session's life cycle events (For example: logout, session destruction).
- Generating the session's life cycle event notifications.
- Implementing session failover facilities.

NOTE The *Sun Java System Access Manager Deployment Guide* contains a detailed section explaining the complete life cycle of a user session.

Session Service Concepts

The following concepts are closely tied together when discussing the Session Service and SSO. To understand the differences between them, consider the following definitions and how they will be used in this chapter.

Session

A *session* is a data structure held in the Access Manager memory that contains session information about an authenticated user.

Session ID

A *session identifier* (ID) is an opaque, globally unique string that programmatically identifies a specific session instance. With the session ID, a resource is able to retrieve session information.

SSOToken

An `SSOToken` is a data structure, defined by the SSO API, that represents a snapshot of the session local to the particular application's memory.

Single Sign-On Process

The next sections describe the process that occurs when a user attempts to gain access to a resource protected by Access Manager.

Contacting A Protected Resource

When a user attempts to access a protected resource via a web browser, a policy agent installed on the server that hosts the resource intercepts the request and, inspects it to see if it contains a Session ID. If none exists, the request is redirected to Access Manager where the Session Service creates a Session for the requesting user. Initially, the session is in an *invalid* state and does not contain user identity information. It does though contain the aforementioned randomly-generated session ID to represent the user's session. Once the session / session ID is created, the Authentication Service sets a cookie with the session ID only and sends it to the client browser. Simultaneously, a login page is generated by the Authentication Service and returned to the user based upon their configured method of authentication (LDAP, RADIUS, etc.).

NOTE For more information on the different methods of authentication, see "Authentication Methods" on page 105 in Chapter 5, "Authentication Service," of this manual.

Providing User Credentials

The user, having received the login page (as well as the session ID) fills in the appropriate credentials based on the type of authentication. After entering their credentials, the data is sent to the authentication provider (LDAP server, RADIUS server, etc.) for verification. Once the provider has successfully verified the credentials, the user is authenticated. The user's specific session information is retrieved (using the session ID) and the session state is set to *valid*. The user can now be redirected to the resource they were attempting to access.

NOTE In reality, the user can only be redirected to the resource if their assigned policy permits it. More information on the Policy Service can be found in Chapter 9, "Policy Management," of this manual.

Cookies and Sessions

A *cookie* is an information packet generated by a web server and passed to a web browser. It maintains information about the user's habits with regards to the web server by which it has been generated. It does not imply that the user is authenticated. Cookies are domain-specific; for example, a cookie generated by DomainA cannot be used in DomainB. Cookies will only be passed to a server in the domain for which the cookie is set. Conversely, servers may only set a cookie in their own domain.

In an Access Manager deployment, the cookie contains the Session ID, an encrypted string generated by the Session Service. With the session ID, a protected resource can get access to the Session where the user's session information is stored. This information is then used for session validation.

NOTE Details on the attributes stored in the session token can also be found in "Authentication Methods" on page 105 in Chapter 5, "Authentication Service," of this manual.

Session Structure

When a user is successfully authenticated they are assigned a valid session. This session contains a number of attributes and properties that define the user's identity and some time-dependent behaviors (for example, the maximum time before the session expires). The following sections detail these attributes.

NOTE The values of most of these attributes and properties are set by services other than the Session Service (primarily, the Authentication Service). The Session Service only provides storage for session information and enforces some of the time-dependent behavior.

Fixed Attributes

The session token contains the following fixed attributes concerning the authenticated user:

- `ID`—This is the Session ID, a randomly-generated session identifier.
- `ClientDomain`—This is the DNS domain in which the client is located.
- `ClientID`—This is the user DN or the application's principal name.

- `Type`—This is the user or application type.
- `State`—This is the state of the session: valid, invalid, destroyed or inactive.
- `maxIdleTime`—This is the maximum time in minutes without activity before the session will expire and the user must reauthenticate.
- `maxSessionTime`—This is the maximum time in minutes before the session expires and the user must reauthenticate.
- `maxCachingTime`—This is the maximum time in minutes before the client contacts Access Manager to refresh cached session information.
- `latestAccessTime`—This is the last time the user has accessed the resource.
- `creationTime`—This is the time at which the session token was set to a valid state.

Protected And Custom Properties

The session token also contains an extensible set of properties that are divided into two subsets: protected (or core) properties and custom properties. Protected properties are set by Access Manager. Custom properties are set remotely by any application that knows the Session ID.

Protected Properties

The current protected properties are:

- `Organization`—This is the DN of the organization to which the user belongs.
- `Principal`—This is the DN of the user.
- `Principals`—This is a list of names to which the user has authenticated. (This property may have more then one value defined as a pipe separated list.)
- `UserId`—This is the user's DN as returned by the module, or in the case of modules other than LDAP or Membership, the user name. (All `Principals` must map to the same user. The UserID is the user DN to which they map.)
- `UserToken`—This is a user name. (All `Principals` must map to the same user. The UserToken is the user name to which they map.)
- `Host`—This is the host name or IP address for the client.
- `authLevel`—This is the highest level to which the user has authenticated.

- `AuthType`—This is a pipe separated list of authentication modules to which the user has authenticated (For example `module1|module2|module3`).
- `Role`—Applicable for role-based authentication only, this is the role to which the user belongs.
- `Service`—Applicable for service-based authentication only, this is the service to which the user belongs.
- `loginURL`—This is the client's login URL.
- `Hostname`—This is the host name of the client.
- `cookieSupport`—This attribute contains a value of true if the client browser supports cookies.
- `authInstant`—This is a string that specifies the time at which the authentication took place.
- `SessionTimedOut`—This attribute contains a value of true if the session has timed out.

Custom Properties

The custom properties currently used are:

- `clientType`—This is the device type of the client browser.
- `Locale`—This is the locale of the client.
- `CharSet`—This is the determined character set for the client.

Cross-Domain Support For SSO

Access Manager supports cross-domain SSO. A user authenticated to Access Manager in one DNS domain can access resources in another, integrated DNS domain. This cross-domain functionality is achieved using the Cross-Domain Controller servlet in Access Manager and Policy Agents installed in web containers. The Controller communicates with the policy agent that resides on servers where the protected resources are kept.

NOTE The Authentication Service handles SSO requests while the Cross-Domain Controller servlet handles cross-domain SSO requests.

Policy Agents

A *policy agent* polices the web container on which a protected resource lives by enforcing a user's assigned policies. They are an integral part of the cross-domain SSO functionality. Two types of policy agents are supported by Access Manager: the web agent and the J2EE/Java agent. The web agent enforces URL-based policy while the J2EE/Java agent enforces J2EE-based security and policy. Both types are available for installation separately from Access Manager and can be downloaded. Additional information can be found in the *Sun Java System Access Manager Web Policy Agents Guide* and *J2EE Policy Agents Guide*. General information on the Policy Service can be found in Chapter 9, "Policy Management," of this manual.

Cross-Domain Controller

The Cross-Domain Controller is a servlet responsible for redirecting user requests. The default URL for it is `http(s)://identity_server_host.domain_name:port/amserver/cdcservlet`. There are three scenarios where the Controller comes into play:

1. If a request for a protected resource contains no session ID, the agent redirects the user to the Controller which, in turn, redirects the user to the appropriate Authentication Service module. Assuming the user is authenticated, this scenario would then follow the path outlined in either Step 2 or Step 3.

NOTE The authentication process itself is discussed in Chapter 5, "Authentication Service," of this manual.

2. If a request for a protected resource already contains a session ID set in a cookie for the same DNS domain in which the resource is deployed, the agent retrieves it and sends an XML/HTTP request to the Naming, Session and Policy Services to retrieve the identity, session and policy information for the requesting user. The user is allowed or denied access to the resource based on this information.
3. If a request for a protected resource does not contain a session ID set in a cookie for the same DNS domain in which the resource is deployed (For example: it carries a session ID set in a different DNS domain from the one in which the Access Manager is deployed), the agent redirects the request to the Controller with a Liberty AuthnRequest in the query string. The Controller then finds the session ID, extracts it from the cookie, places it in a Liberty AuthnResponse and sends it back to the agent. The agent finds the session ID, extracts it from

the AuthnResponse, sets it in a cookie for the new domain, and sends an XML/HTTP request to the Naming, Session and Policy Services to retrieve the identity, session and policy information for the requesting user. The user is allowed or denied access to the resource based on this information.

NOTE The Liberty AuthnRequest and AuthnResponse are part of the Federation Management module. For more information, see the Access Manager *Federation Management Guide*.

A Cross-Domain SSO Scenario

In one scenario, the Access Manager instance for DomainA is its authentication provider. A user authenticates to Access Manager in DomainA and, after authentication, the session is set for DomainA. ServerB, on the other hand, is protected by a policy agent talking to an Access Manager in DomainB.

NOTE This is just one scenario; it is not obligatory to have an installed instance of Access Manager in both domains to use the cross-domain feature.

The Access Manager instance in DomainB recognizes the DomainA instance as an authentication provider. If UserA, after authenticating to DomainA, requests a resource on ServerB, the policy agent for DomainB checks for a session ID and will find that there is none (authorizing access to DomainB, that is). The agent then redirects the request to the Cross-Domain Controller running with the Access Manager instance in DomainB. The servlet, following the path outlined in Step 3 on page 79, finds the session ID from DomainA, extracts it from the cookie, places it in a Liberty AuthnResponse and sends it back to the agent. The agent finds the session ID and sets a cookie for DomainB using the session ID. The agent then sends an XML/HTTP request to the Naming, Session and Policy Services deployed in DomainB. Since the instance of Access Manager in DomianB recognizes the instance of Access Manager in DomainA as an authentication provider, DomainB retrieves identity, session and policy information for the requesting user from DomainA. The user is then allowed or denied access to the resource based on this information.

NOTE Access Manager uses a combination of URL parameters and cookies to implement cross-domain SSO. If a cookie is set in DomainA, the cookie value is carried over to DomainB using the URL parameters, and a new cookie can be set for DomainB with the same cookie name and value.

Enabling Cross-Domain Single Sign-On

As described, in order to exchange session information across two different domains, Policy Agents and the Cross-Domain Controller communicate with each other. By default, Access Manager is installed with the servlet. Policy agents, on the other hand, are installed separately. When installing the agent, the option to configure it for CDSSO must be selected. The cookie domain for the agent must be configured after installation. This is done by editing the `AMAgent.properties` file. The `com.sun.am.policy.agents.cookieDomainList` property must be set with the domain in which the agent is installed. If the field is left blank, the cookie domain will be set to the FQDN of the web server on which the agent is installed. Additional information on enabling cross-domain single sign-on can be found in the *Web Policy Agents Guide* and the *J2EE Policy Agents Guide*.

SSO API

The Session Service provides Java and C API to allow external applications to participate in the SSO functionality. All Access Manager services (except for Authentication) require a valid session (programmatically referred to as `SSOToken`) to process a HTTP request. External applications wishing to use the SSO functionality must also use the `SSOToken` to authenticate the user's identity. With the SSO API, an external application can retrieve it and, in turn, the user's identity, session and policy information. The application then uses this information to determine whether to provide user access to a protected resource.

After successfully authenticating to Access Manager, a user carries their Session ID with them using browser cookies or URL query parameters. Now, each time a user requests access to a protected application, the application needs to verify their identity. Assume a user authenticates to `http://www.orgA.com/Store` successfully and later tries to access `http://www.orgA.com/UpdateInfo`, a service that is *SSO-enabled*. Rather than having the second application authenticate the user again, it can use the API and the user's session to determine if the user is already authenticated. If the methods determine that the user has already been authenticated (and the session is still valid), access to this page can be achieved. Otherwise, the user would be prompted to authenticate again. The SSO API can also be used to create or destroy a `SSOToken`, or to listen for `SSOToken` events. (An *event* might be a `SSOToken` timing out because the user has reached the their maximum time limit.) Following are both the Java API Overview and C API Overview.

Java API Overview

In Java, the main classes of the SSO API are `SSOTokenManager`, `SSOToken` and `SSOTokenListener`. The `SSOTokenManager` class is used to get, destroy, validate, and refresh a session token which is represented by the `SSOToken` class. The `SSOTokenListener` class allows the application to be notified when a `SSOToken` has become invalid, for example when a session has timed out.

SSOTokenManager Class

The `SSOTokenManager` class contains the methods needed to get, validate, destroy and refresh session tokens. `SSOTokenManager` is implemented using the singleton design pattern. In order to obtain an instance of `SSOTokenManager`, the `SSOTokenManager.getInstance()` method must be called. An instance of `SSOTokenManager` can then be used to instantiate an `SSOToken` object using one of the overloaded forms of the `createSSOToken()` method.

The `destroyToken()` method would be called to invalidate and delete a token when its session has ended. The `isValidToken()` and `validateToken()` methods can be called to verify whether a token is valid, or authenticated. `isValidToken()` returns true or false depending on whether the token is valid or invalid, respectively. `validateToken()` throws an exception only when the token is invalid; nothing happens if the token is valid. The `refreshSession()` method resets the idle time of the session. Code Example 4-1 illustrates one way in which the `SSOTokenManager` class can be used.

Code Example 4-1 Sample Uses Of SSOTokenManager Code

```
try {
        /* get an instance of the SSOTokenManager */
        SSOTokenManager ssoManager = SSOTokenManager.getInstance();

        /* The request here is the HttpServletRequest. Get
        /* SSOToken for session associated with this request. */
        SSOToken ssoToken = ssoManager.createSSOToken(request);

        /* use isValid method to check if token is valid or not.
         * This method returns true for valid token, false otherwise. */
        if (ssoManager.isValidToken(ssoToken)) {
          /* If token is valid, this information may be enough for
           * some applications to grant access to the requested
           * resource. A valid user represents a user who is
           * already authenticated. An application can further
           * utilize user identity information to apply
           * personalization logic.
           */
        } else {
           /* Token is not valid, redirect the user login page. */
        }
```

Code Example 4-1 Sample Uses Of SSOTokenManager Code *(Continued)*

```
        /* Alternative: use of validateToken method to check
           * if token is valid */
        try {
          ssoManager.validateToken(ssoToken);
          /* handle token is valid */
        } catch (SSOException e) {
          /* handle token is invalid */
        }

        /*refresh session. idle time should be 0 after refresh. */
        ssoManager.refreshSession(ssoToken);

      } catch (SSOException e) {
        /* An error has occurred. Do error handling here. */
      }
```

SSOTokenID Interface

The `SSOTokenID` interface is used to identify the `SSOToken` object.

CAUTION The string value of `SSOTokenID` is globally unique and must only be known to the client browser, Access Manager and the application code. Exposing it to unauthorized users or applications can lead to a security breach by allowing a malicious attacker to impersonate a user.

SSOToken Interface

The `SSOToken` interface represents a *single sign-on* token returned from the `SSOTokenManager.createSSOToken()` method, and contains information such as the authenticated principal name, authentication method, and session information (session idle time, maximum session time, etc.). The `SSOToken` interface has methods to get predefined session information, such as `getAuthType()` for the authentication type, as well as a method `getProperty()` to get any information about the session, predefined or otherwise (for example, information set by the application). The method `setProperty()` can be used by the application to set application-specific information in the session. The `addSSOTokenListener()` method can be used to set a listener to be invoked when the session state has become invalid.

CAUTION The methods `getTimeLeft()` and `getIdleTime()` return values in seconds while the methods `getMaxSessionTime()` and `getMaxIdleTime()` return values in minutes.

Code Example 4-2 shows an example of `SSOToken` code.

Code Example 4-2 Sample Use Of SSOToken

```
/* get http request output stream for output */
      ServletOutputStream out = response.getOutputStream();

      /* get the sso token from http request */
      SSOTokenManager ssoManager = SSOTokenManager.getInstance();
      SSOToken ssoToken = ssoManager.createSSOToken(request);

      /* get the sso token ID from the sso token */
      SSOTokenID ssoTokenID = ssoToken.getTokenID();
      out.println("The SSO Token ID is "+ssoTokenID.toString());

      /* use validate method to check if the token is valid */
      try {
        ssoManager.validateToken(ssoToken);
        out.println("The SSO Token validated.");
      } catch (SSOException e) {
        out.println("The SSO Token failed to validate.");
      }

      /* use isValid method to check if the token is valid */
      if (!ssoManager.isValidToken(token)) {
        out.println("The SSO Token is not valid.");
      } else {
        /* get some values from the SSO Token */
        java.security.Principal principal = ssoToken.getPrincipal();
          out.println("Principal name is "+principal.getName());
        String authType = ssoToken.getAuthType();
          out.println("Authentication type is "+authType);
          int authLevel = ssoToken.getAuthLevel();
          out.println("Authentication level is "+authLevel);
        long idleTime = ssoToken.getIdleTime();
          out.println("Idle time is "+idleTime);
          long maxIdleTime = ssoToken.getMaxIdleTime();
          out.println("Max idle time is "+maxIdleTime);
          long maxTime = token.getMaxSessionTime();
          out.println("Max session time is "+maxTime);
          String host = ssoToken.getHostName();
          out.println("Host name is "+host);
          /* host name is a predefined information of the session,
          /* and can also be obtained the following way */
          String hostProperty = ssoToken.getProperty("HOST");
          out.println("Host property is "+hostProperty);
          /* set application specific information in session */
          String appPropertyName = "appProperty";
          String appPropertyValue = "appValue";
          ssoToken.setProperty(appPropertyName, appPropertyValue);
          /* now get the app specific information back */
          String appValue = ssoToken.getProperty(appPropertyName);
          if (appValue.equals(appPropertyValue)) {
              out.println("Property "+appPropertyName+",  value
"+appPropertyValue+" verified to be set.");
```

Code Example 4-2 Sample Use Of SSOToken *(Continued)*

```
          } else {
              out.println("ALERT: Setting property "+appPropertyName+"
failed!");
          }
      }
```

A code sample using the `getTokenID` method is illustrated in Code Example 4-3. With this code, a cookie is created from an `SSOToken` in order to make SSO work for protected resources not residing on the same server as Access Manager.

Code Example 4-3 Sample Code To Create A Cookie From Session Token

```
// Get SSOToken string
String strToken = null;
strToken = getSSOToken().getTokenID().toString();
// Set it to response as cookies
String s = strToken;
String ssotokencookiename = "iPlanetDirectoryPro";
String ssotokencookiedomain = ".mydomain.com.tw";
String ssotokencookiepath = "/";
String gt = "/welcomepage.jsp";
Cookie cookie = new Cookie(ssotokencookiename,s);
cookie.setDomain(ssotokencookiedomain);
cookie.setPath(ssotokencookiepath);
response.addCookie(cookie);
response.sendRedirect(gt);
```

SSOTokenEvent

The `SSOTokenEvent` interface represents a token event. An event is, for example, when a session has been idle for over a maximum idle time limit, or when a session has reached its maximum allowed time.

SSOTokenListener

The `SSOTokenListener` interface represents a token notification object. An implementation of the `SSOTokenListener` interface must be written, then registered with the `SSOTokenManager` to be invoked when a token event occurs.

The SSOTokenListener interface provides a mechanism to notify applications when a session token has become invalid due to, for instance, the session reaching maximum idle time or the maximum session time. Applications wishing to be notified must write an implementation of the SSOTokenListener interface, then register the implementation through the SSOToken.addSSOTokenListener method. When the SSOToken state has become invalid, the SSOTokenListener implementation's ssoTokenChanged method will be invoked with a SSOTokenEvent object containing the event type, time, and SSOToken object with the new SSOToken state and other properties of the SSOToken.

Code Example 4-4 Sample Code For SSOToken Event And SSOToken Listener

```
public class SampleTokenListener implements SSOTokenListener {
      public void ssoTokenChanged(SSOTokenEvent event) {
        try {
          SSOToken token = event.getToken();
          int type = event.getType();
          long time = event.getTime();
          SSOTokenID id = token.getTokenID();
          System.out.println("Token id: " + id.toString() + "is not valid
anymore");
          /* redirect user to login */
          ......
        } catch (Exception e) {
              System.out.println(e.getMessage());
        }
      }
      }

      public SampleTestRoutine {
        ...
        SSOTokenManager ssoManager = SSOTokenManager.getInstance();
        SSOToken ssoToken = SSOManager.createSSOToken(request);
        SSOTokenListener sampleListener = new SampleTokenListener();
        ssoToken.addSSOTokenListener(sampleListener);
      ...
      }
```

Sample SSO Java Files

Access Manager provides three groups of sample Java files. With these samples, a developer can create a session token in several ways:

1. With the SSO Servlet Sample, a session token can be created for an application that runs on the Access Manager server.
2. With the Remote SSO Sample, a session token can be created for an application that runs on a server other than the Access Manager server.

3. With the Command Line SSO Sample, a session token can be created by a session ID string and passed through the command line.

The sample files are located in the *IdentityServer_base*/`SUNWam/samples/sso` directory.

SSO Servlet Sample

This sample can be used to create a token for an application that resides on the same server as the Access Manager application. The files used for this sample are:

- `Readme.html`
- `SampleTokenListener.java`
- `SSOTokenSampleServlet.java`

The instructions in `Readme.html` can be followed to run this code.

Remote SSO Sample

This sample can be used to create a token for an application that resides on a different server from the one on which the Access Manager application lives. The files used for this sample are:

- `remote.html`
- `SSOTokenFromRemoteServlet.java`
- `SSOTokenSampleServlet.java`

The instructions in `remote.html` can be followed to run this code.

Command Line SSO Sample

This sample illustrates how to validate a user from the command line using a session ID string. The files used for this sample are:

- `ssocli.txt`
- `CommandLineSSO.java`
- `SSOTokenSample.java`

The instructions in `ssocli.txt` can be followed to run this code.

C API Overview

The C API are provided in the `SUNWamcom` package which comes with Access Manager or any of its downloadable agents. The package includes header files, libraries and samples.

CAUTION Previous releases of Access Manager contained C libraries in *IdentityServer_base*/`lib/capi`. The `capi` directory is being deprectated, and is curently available for backward compatability. It will be removed in the next release, and therefore it is highly recommended that existing application paths to this directory are changed and new applications do not access it. Paths include `RPATH`, `LD_LIBRARY_PATH`, `PATH`, compiler options, etc.)

C SSO Include Files

Include files for the C SSO API are `am_sso.h` and `am_notify.h`. `am_sso.h` must be included for any SSO routines. `am_notify.h` must be included for parsing notification messages from the server and calling SSO listeners.

C SSO Properties

Certain properties must be read in and passed to `am_sso_init()`, the routine which initializes C API. Because of this, `am_sso_init()` must be called before any other SSO interface. The default properties file used is `AMAgent.properties`, located in *IdentityServer_base*/`SUNWam/config/`. The following properties must be set:

- The `com.sun.am.namingURL` property specifies the URL for the Naming Service. This service is used to find the URL of the Session Service for the given `SSOToken` ID. This property must be set as:

  ```
  com.sun.am.namingURL =
  https://myhost.mydomain.com:58080/amserver/namingservice
  ```

- The `com.sun.am.notificationEnabled` and `com.sun.am.notificationURL` properties specify whether notification is enabled, and if enabled, a URL where the application can listen for messages from Access Manager. These properties must be set as:

  ```
  com.sun.am.notificationEnabled=true
  ```

NOTE If `com.sun.am.notificationEnabled` is not found in the properties file, the default is false.

```
com.sun.am.notificationURL=https://myhost.mydomain.com:8000/myU
RL
```

- The `com.sun.am.sso.cacheEntryLifeTime` property specifies how long, in minutes, a session token can live in cache before it should be removed. This property must be set as:

  ```
  com.sun.am.sso.cacheEntryLifeTime=5
  ```

  If not set, the default is 3 minutes.

- The `com.sun.am.sso.checkCacheInterval` property specifies how often, in minutes, the cache should be checked for entries that have reached the cache entry life time. This property must be set as:

  ```
  com.sun.am.sso.checkCacheInterval=5
  ```

- The `com.sun.am.sso.maxThreads` specify the maximum number threads the SSO API should invoke for handling notifications. The API maintains a thread pool and invokes a thread for each notification. If the maximum number of threads has been reached, the notification will wait until a thread is available. If not specified the default maximum number of threads is 10. This property must be set as:

  ```
  com.sun.am.sso.maxThreads = 5
  ```

- The `com.sun.am.cookieEnabled` property specifies whether the session ID found in the cookie is URL encoded. If true, it will be URL decoded before sent to Access Manager for any session operation. This property must be set as:

  ```
  com.sun.am.cookieEncoded = true|false
  ```

More information on properties in the `AMAgent.properties` file can be found in the *Web Policy Agents Guide* and the *J2EE Policy Agents Guide*.

C SSO interfaces

The C SSO interfaces consist of the following routines. A detailed description of the input and output parameters for each interface is in the header files.

- Initialization and Cleanup
- Get, Validate, Refresh And Destroy SSO Token
- Get Session Information Interfaces

- Get And Set Property Interfaces
- Listener And Notify Interfaces

Initialization and Cleanup

To use the C SSO API, the `am_sso_init()` routine needs to be called before any other routines. This interface initializes the internal SSO module. At the end of all SSO routines, `am_cleanup()` should be called to cleanup the internal SSO module. Code Example 4-5 on page 90 is a code sample for these interfaces.

am_sso_init() initializes internal data structures for talking to the Session Service. It takes a properties input parameter that contains name /value pairs from a configuration or properties file, and returns a status on the success or failure of the initialization. The properties used by the C SSO API are covered in "C SSO Properties" on page 88.

am_cleanup() cleans up all internal data structures created by `am_sso_init`, `am_auth_init`, or `am_policy_init`. `am_cleanup()` needs to be called only once when using any of the Access Manager C API interfaces (authentication, SSO or policy).

Code Example 4-5 Code Sample For am_sso_init and am_cleanup

```
#include <am_sso.h>

int main() {
   am_properties_t *properties;
   am_status_t status;

   /* create a properties handle */
   status = am_properties_create(&properties);
   if (status != AM_SUCCESS) {
        printf("am_properties_create failed.\n");
        exit(1);
   }

   /* load properties from a properties file */
   status = am_properties_load(properties, "./myPropertiesFile");
   if (status != AM_SUCCESS) {
       printf("am_properties_load failed.\n");
       exit(1);
   }

   /* initialize SSO module */
   status = am_sso_init(properties);
   if (status != AM_SUCCESS) {
       printf("am_sso_init failed.\n");
       return 1;
   }

    /* login through auth module, and do auth functions.
```

Code Example 4-5 Code Sample For am_sso_init and am_cleanup *(Continued)*

```
          * ...
          */

        /* do sso functions
         * ...
         */

         /* done - cleanup. */
         status = am_cleanup();
         if (status != AM_SUCCESS) {
             printf("am_cleanup failed!\n");
             return 1;
         }
         /* free memory for properties */
         status = am_properties_destroy(properties);
         if (status != AM_SUCCESS) {
             printf("Failed to free properties.\n");
             return 1;
         }

         /* exit program successfully. */
         return 0;
     }
```

Get, Validate, Refresh And Destroy SSO Token

A user needs to be authenticated to get the token ID for their login session. A token can be obtained with the token ID and the `am_sso_create_sso_token_handle` interface. This interface checks to see if the token is in its local cache and, if not, goes to the server to get the session information associated with the token ID and caches it. If the reset flag is set to `true`, this interface will refresh the idle time of the token on the server. Here is the interface of `am_sso_create_sso_token_handle`:

- `am_status_t am_sso_create_sso_token_handle(am_sso_token_handle_t * sso_token_handle_ptr, const char *sso_token_id, boolean_t refresh_token);`

Once a token handle is obtained, the caller can check if the session is valid with the `am_sso_is_valid_token` interface. The `am_sso_token_validate` interface will flush the token handle in the local cache (if any) and go to the server to fetch the latest session information. The `am_sso_refresh_token` will also flush the token handle in the local cache (if any) and go to the server to fetch the session information. In addition, it will reset the idle time of the session on the server. Here are the token-related interfaces:

- `boolean_t    am_sso_is_valid_token(am_sso_token_handle_t sso_token_handle);`
- `am_status_t am_sso_validate_token(am_sso_token_handle_t sso_token_handle);`
- `am_status_t am_sso_refresh_token(am_sso_token_handle_t sso_token_handle);`

When caller is done with a token handle, it must be freed by calling `am_sso_destroy_sso_token_handle` to prevent memory leak. Here is that interface:

- am_status_t am_sso_destroy_sso_token_handle(am_sso_token_handle_t sso_token_handle);

The session associated with the token can be invalidated or ended with `am_sso_invalidate_token`. Although this ends the session for the user, the proper way to log out is through `am_auth_logout`. Using the former interface to end a session will result in authentication resources associated with the session to remain on the server unnecessarily until the session has timed out. Here is the interface for `am_sso_invalidate_token`:

- `am_status_t am_sso_invalidate_token(am_sso_token_handle_t sso_token_handle);`

Get Session Information Interfaces

The following interfaces make it convenient to get server-defined information (or properties) about the session associated with a token. This can include the session idle time, max session time, etc.

- `const char * am_sso_get_sso_token_id(const am_sso_token_handle_t sso_token_handle);`
- `const char * am_sso_get_auth_type(const am_sso_token_handle_t sso_token_handle);`
- `unsigned long am_sso_get_auth_level(const am_sso_token_handle_t sso_token_handle);`
- `time_t          am_sso_get_idle_time(const am_sso_token_handle_t sso_token_handle);`
- `time_t          am_sso_get_max_idle_time(const am_sso_token_handle_t sso_token_handle);`
- `time_t          am_sso_get_time_left(const am_sso_token_handle_t sso_token_handle);`

- `time_t          am_sso_get_max_session_time(const am_sso_token_handle_t sso_token_handle);`
- `const char * am_sso_get_principal(const am_sso_token_handle_t sso_token_handle);`
- `am_string_set_t am_sso_get_principal_set(const am_sso_token_handle_t sso_token_handle);`
- `const char * am_sso_get_host(const am_sso_token_handle_t sso_token_handle);`

Get And Set Property Interfaces

The get and set property interfaces allows an application to get any property (server or application defined) and to set any property in a session. Note that `am_sso_set_property` will update the `sso_token_handle` with the latest session properties from Access Manager, including the new property that was set. In addition, if the property that is given in `prop_name` is a protected property, `am_sso_set_property` will return success, however the value given will not be set as it is a property protected by Access Manager. These interfaces are:

- `const char * am_sso_get_property(const am_sso_token_handle_t sso_token_handle, const char *prop_name);`
- `am_status_t am_sso_set_property(am_sso_token_handle_t sso_token_handle, const char *prop_name, const char *prop_value);`

Code Example 4-6 is a sample of the SSO get, set, create, refresh, validate, invalidate, and destroy interfaces.

Code Example 4-6 Sample Code For Get, Set, Create, Refresh, Validate, Invalidate, and Destroy Interfaces

```
     /* initialize sso as in previous sample */

     am_status_t status = NULL;
     am_sso_token_handle_t sso_handle = NULL;
     char *session_status = NULL;
     am_string_set_t principal_set = NULL;

     /* create sso token handle */
     status = am_sso_create_sso_token_handle(&sso_handle, sso_token_id,
false);
     if (status != AM_SUCCESS) {
         printf("Failed getting sso token handle for sso token id %s.\n",
sso_token_id);
         return 1;
```

Code Example 4-6 Sample Code For Get, Set, Create, Refresh, Validate, Invalidate, and Destroy Interfaces *(Continued)*

```
     }

     /* check if session is valid */
     session_status = am_sso_is_valid_token(sso_handle) ? "Valid" :
"Invalid";
     printf("Session state is %s\n", session_status);

     /* check if session is valid using validate. This also updates the
handle with info from the server */
     status = am_sso_validate_token(sso_handle);
     if (status == AM_SUCCESS) {
         printf("Session state is valid.\n");
     } else if (status == AM_INVALID_SESSION) {
         printf("Session status is invalid.\n");
     } else {
         printf("Error validating sso token.\n");
         return 1;
     }

     /* get info on the session */
     printf("SSO Token ID is %s.\n", am_sso_get_sso_token_id(sso_handle));
     printf("Auth type is %s.\n", am_sso_get_auth_type(sso_handle));
     printf("Auth level is %d.\n", am_sso_get_auth_level(sso_handle));
     printf("Idle time is %d.\n", am_sso_get_idle_time(sso_handle));
     printf("Max Idle time is %d.\n", am_sso_get_max_idle_time(sso_handle));
     printf("Time left is %d.\n", am_sso_get_time_left(sso_handle));
     printf("Max session time is %d.\n",
am_sso_get_max_session_time(sso_handle));
     printf("Principal is %s.\n", am_sso_get_principal(sso_handle));
     printf("Host is %s.\n", am_sso_get_host(sso_handle));
     principal_set = am_sso_get_principal_set(sso_handle);
     if (principal_set == NULL) {
            printf("ERROR: Principal set is NULL!\n");
     }else {
            printf("Principal set size %d.\n", principal_set->size);
            for (i = 0; i < principal_set->size; i++) {
                printf("Principal[%d] = %s.\n", i,
principal_set->strings[i]);
            }
            am_string_set_destroy(principal_set);
     }

     /* get "HOST" property on the session. Same as am_sso_get_host(). */
     printf("Host is %s.\n", am_sso_get_property(sso_handle, "HOST"));

     /* set a application defined property and get it back */
     status = am_sso_set_property(sso_handle, "AppPropName",
"AppPropValue");
     if (status != AM_SUCCESS) {
         printf("Error setting property.\n");
         return 1;
     }
```

Code Example 4-6 Sample Code For Get, Set, Create, Refresh, Validate, Invalidate, and Destroy Interfaces *(Continued)*

```
     printf("AppPropName value is %s.\n", am_sso_get_property(sso_handle,
"AppPropName");

     /* refresh token, idle time should be 0 after refresh */
     status = am_sso_refresh_token(sso_handle);
     if (status != AM_SUCCESS) {
         printf("Error refreshing token !\n");
         return 1;
     }
     printf("After refresh, idle time is %d.\n",
am_sso_get_idle_time(sso_handle));

     /* end this session abruptly. am_auth_logout() is the right way to end
session */
     status = am_sso_invalidate_token(sso_handle);
     if (status != AM_SUCCESS) {
         printf("Error invalidating token.\n");
         return 1;
     }

     /* we're done with sso token handle. free memory for sso handle. */
     status = am_sso_destroy_sso_token_handle(sso_handle);
     if (status != AM_SUCCESS) {
         printf("Failed to free sso token handle.\n");
         return 1;
     }

     /* call am_cleanup, and other cleanup routines as in previous sample */
```

Listener And Notify Interfaces

Applications can be notified when a session has become invalid, possibly because it has been idle over a time limit, or it has reached the maximum session time. This is done by implementing a listener function of type `am_sso_token_listener_func_t`, which takes a SSO token handle, event type, event time, application-defined arguments handle, and a boolean argument to indicate whether the listener function should be called in the calling thread or dispatched to a thread from the internal thread pool managed by the C SDK. This listener function must be registered to be invoked when the session has ended and notification must be enabled for an application to receive notifications. Notification is enabled by setting the property `com.sun.am.notificationEnabled` to true, and by providing a URL where the application is receiving HTTP messages from Access Manager. The URL where the application is receiving messages from the Access Manager is expected to take any message from the server (as an XML string) and pass it to `am_notify()`. `am_notify()` will parse the message and invoke session listeners or policy listeners depending on whether the message is a session or policy notification. Code Example 4-7 is a sample implementation of SSOToken listener and how to register it.

Code Example 4-7 Sample Implementation Of SSOToken Listener

```
     void sample_listener_func(
                    am_sso_token_handle_t sso_token_handle,
                    const am_sso_token_event_type_t event_type,
                    const time_t event_time,
                    void *opaque)
     {
         if (sso_token_handle != NULL) {
             const char *sso_token_id =
am_sso_get_sso_token_id(sso_token_handle);
             boolean_t is_valid = am_sso_is_valid_token(sso_token_handle);
             printf("sso token id is %s.\n",
                    sso_token_id==NULL?"NULL":sso_token_id);
             printf("session state is %s.\n",
                      is_valid == B_TRUE ? "valid":"invalid");
             printf("event type %d.\n", event_type);
             printf("event time %d.\n", event_time);
         }
         else {
             printf("Error: sso token handle is null!");
         }
         if (opaque)
             *(int *)opaque = 1;
         return;
     }

     int main(int argc, char *argv[]) {

     am_status_t status;
     char *sso_token_id = argv[1];
     int listener_func_done = 0;

     /* initialize sso as in previous samples */

     /* get sso token handle */
     status = am_sso_create_sso_token_handle(&sso_handle, sso_token_id,
false);

     /* register listener function. notification must be enabled, if not,
status AM_NOTIF_NOT_ENABLED will be returned. */
 status = am_sso_add_sso_token_listener(sso_handle, sample_listener_func,
&listener_func_done, B_TRUE);
     if (status != AM_SUCCESS) {
         printf("Failed to register sample listener function.\n");
         return 1;
     }
```

C SSO Sample

A sample for the C SSO API is provided in the `SUNWamcom` package. The README file in the samples directory contains information on each sample including compile instructions and how to run the samples for testing. The sample for C SSO is `am_sso_test.c`. The usage is `am_sso_test -u [user] -p [password] [-f properties file] [-l logfile]`. Access Manager must be available with LDAP authentication to test the sample. See the `README` file and the sample itself for more information.

Java versus C API

The following table provides a side by side comparison of the Java and C SSO API.

Table 4-1 Comparison Between Java And C SSO API

Java Interface	C Interface
SSOTokenManager	am_status_t
SSOTokenManager.getInstance()	am_sso_init(am_properties_t properties)
SSOToken	am_status_t
SSOTokenManager.createSSOToken(String tokenId)	am_sso_create_sso_token_handle(am_sso_token_handle_t *sso_token_handle_ptr, const char *sso_token_id, am_bool_t reset_idle_timer)
boolean	boolean_t
SSOTokenManager.isValidToken(SSOToken token)	am_sso_is_valid_token(const am_sso_token_handle_t sso_token_handle)
void	am_status_t
SSOTokenManager.validateToken(SSOToken token)	am_sso_validate_token(const am_sso_token_handle_t sso_token_handle)
void	am_status_t
SSOTokenManager.destroyToken(SSOToken token)	am_sso_invalidate_token(const am_sso_token_handle_t sso_token_handle)
void	am_status_t
SSOTokenManager.refreshSession(SSOToken token)	am_sso_refresh_session(am_sso_token_handle_t sso_token_handle)
Principal	char *
SSOToken.getPrincipal()	am_sso_get_principal(const am_sso_token_handle_t sso_token_handle)

Table 4-1 Comparison Between Java And C SSO API *(Continued)*

Java Interface	**C Interface**
int	unsigned long
SSOToken.getAuthLevel()	am_sso_get_auth_level(const am_sso_token_handle_t sso_token_handle)
String	char *
SSOToken.getAuthType()	am_sso_get_auth_type(const am_sso_token_handle_t sso_token_handle)
String	char *
SSOToken.getHostName()	am_sso_get_host(const am_sso_token_handle_t sso_token_handle)
long	time_t
SSOToken.getIdleTime()	am_sso_get_max_idle_time(const am_sso_token_handle_t sso_token_handle)
long	time_t
SSOToken.getMaxIdleTime()	am_sso_get_max_idle_time(const am_sso_token_handle_t sso_token_handle)
SSOTokenID	char *
SSOToken.getTokenID()	am_sso_get_sso_token_id(const am_sso_token_handle_t sso_token_handle)
String	char *
SSOToken. getProperty(java.lang.String name)	am_sso_get_property(const am_sso_token_handle_t sso_token_handle, const char *property_name)
void	am_status_t
SSOToken.setProperty(String name, String value)	am_sso_set_property(am_sso_token_handle_t sso_token_handle, const char *name, const char *value)
void	am_status_t
SSOToken.addSSOTokenListener(SSOTokenListener listener)	am_sso_add_sso_token_listener(am_sso_token_handle_t sso_token_handle, const am_sso_token_listener_func_t listener, void *args, boolean_t dispatch_in_sep_thread)
String	am_status_t
SSOToken.getProperty("principals");	am_sso_get_principal_set(am_sso_token_handle_t sso_handle)
N/A	am_status_t
	am_sso_destroy_sso_token_handle(am_sso_token_handle_t sso_handle)

Table 4-1 Comparison Between Java And C SSO API *(Continued)*

Java Interface	C Interface
N/A	void am_cleanup()

Non-Web-Based Applications

Access Manager provides the SSO API primarily for web-based applications, although it can be extended to any non-web-based applications with limitations. With non-web-based applications, their are two possible ways to use the API.

1. The application has to obtain the Access Manager cookie value and pass it into the SSO client methods to get to the session token. The method used for this process is application-specific.
2. Command line applications, such as `amadmin`, can be used. In this case, session tokens can be created to access the Directory Server directly. There is no session created, making the Access Manager access valid only within that process or VM.

SSO Samples

Access Manager provides the files necessary to compile and run a sample SSO application. There are three ways in which this can be done:

- Compiling and running a SSO application local to Access Manager.
- Installing and running the SSO SDK from a remote client.
- Running the SSO application from the command line.

More specific information on these samples can be found in "Sample SSO Java Files" on page 86.

Chapter 5

Customizing the Authentication User Interface

The authentication service provides the web-based Graphical User Interface (GUI) for all out-of-box and custom authentication modules installed in the Sun Java™ System Access Manager 6 2005Q1 deployment. This interface provides a dynamic and customizable means for gathering authentication credentials by presenting the web-based login requirement pages to a user requesting access.

The authentication service GUI is built on top of JATO (J2EE Assisted Take-Off), a Java 2 Enterprise Edition (J2EE) presentation application framework. This framework is used to help developers build complete functional Web applications.

The following topics are covered in this chapter:

- User Interface Files You Can Modify
- Customizing Branding and Functionality
- Customizing the Self-Registration Page

User Interface Files You Can Modify

The authentication GUI dynamically displays the required credentials information depending upon the authentication module invoked at run time. The Table 5-1 lists the types of files you can modify to convey custom representations of Login pages, Logout pages, and error messages. Detailed information is provided in following sections.

Table 5-1 Authentication User Interface Files and Their Locations at Installation

File Type	Default Location
services.war File	*AccessManager-base*/SUNWam/web-src/services
Java Server Pages	*AccessManager-base*/SUNWam/web-src/services/config/auth/default
XML Files	*AccessManager-base*/SUNWam/web-src/services/config/auth/default
JavaScript Files	*AccessManager-base*/SUNWam/web-src/services/js
Cascading Style Sheets	<*AccessManager-base*/SUNWam/web-src/services/css
Images	*AccessManager-base*/SUNWam/web-src/services/login_images
Localization Files	*AccessManager-base*/SUNWam/locale

To access the default Login page, use the following URL:

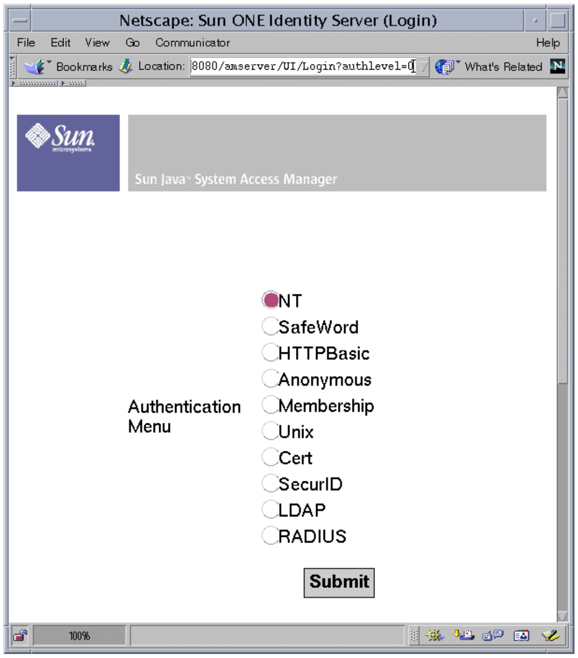
```
<server_protocol>://<server_host>.<server_domain>:<server_port>/
    <service_deploy_uri>/UI/Login
```

To access the default Logout page, use the following URL:

```
<server_protocol>://<server_host>.<server_domain>:<server_port>/
    <service_deploy_uri>/UI/Logout
```

The following image illustrates the first page seen for a login when all modules have been configured for authlevel 0.

Figure 5-1 Default Login Page when authlevel=0


Netscape: Sun ONE Identity Server (Login)
File Edit View Go Communicator Help
Bookmarks Location: 8080/amserver/UI/Login?authlevel=0
What's Related
Sun microsystems
Sun Java System Access Manager
Authentication Menu
NT
SafeWord
HTTPBasic
Anonymous
Membership
Unix
Cert
SecurID
LDAP
RADIUS
Submit
100%

Authentication Menu: [✓ NT] SafeWord HTTPBasic Anonymous Membership Unix Cert SecurID LDAP RADIUS

services.war File

The `services.war` contains all the files you need to modify the authentication GUI. When you install Access Manager on Sun ONE Application Server, on Sun Java ES Web Server, or on WebLogic Web Server, `services.war` is automatically installed and deployed. Its files and directories are installed by default in the following location:

AccessManager-base`/SUNWam/web-src/services`

If you install Access Manager on other web containers, you may have to manually deploy `services.war`. See the documentation that comes with the web container.

Once you've modified the authentication GUI files, in order to see the changes in the actual GUI, you must update and then redeploy `services.war`. See "Updating and Redeploying services.war" on page 117 in this chapter for instructions. See Appendix C, "WAR Files" on page 359 for general information on updating and redeploying Access Manager `.war` files.

Java Server Pages

All authentication GUI pages are `.jsp` files with embedded JATO tags. You do not need to understand JATO to customize Access Manager GUI pages. Java server pages handle both the UI elements and disciplines displayed through peer ViewBeans. By default, JSP pages are installed in the following directory: *AccessManager-base*`/SUNWam/web-src/services/config/auth/default`

Note that Java server pages are looked up from the deployed location. In previous Access Manager versions, the Java server pages were looked up from the installed location.

Customizing the Login Page

The Login page is a common Login page used by most authentication modules except for the Membership module. For all other modules, at run time the Login page dynamically displays all necessary GUI elements for the required credentials. For example, the LDAP authentication module Login page dynamically displays the LDAP module header, LDAP User name, and Password fields.

You can customize the following Login page UI elements:

- Module Header text
- User Name label and field
- Password label and field
- Choice value label and field
  Note, the field is a radio button by default, but can be change to a check box.
- Image (at the module level)
- Login button

Customizing JSP Templates

Use the JSP templates to customize the look and feel of presented in the graphical user interface (GUI). See "To Modify Branding and Functionality" on page 113 for detailed instructions. Table 5-2 contains provides descriptions of templates you can customize. The templates are located in the following directory:

IdentityServer_base`/SUNWam/web-src/services/config/auth/default`

Table 5-2 List of Customizable JSP Templates

File Name	Purpose
`account_expired.jsp`	Informs the user that their account has expired and should contact the system administrator.
`auth_error_template.jsp`	Informs the user when an internal authentication error has occurred. This usually indicates an authentication service configuration issue.
`authException.jsp`	Informs the user that an error has occurred during authentication.
`disclaimer.jsp`	This is a customizable disclaimer page used in the Self-registration authentication module.
`Exception.jsp`	Informs the user that an error has occurred.
`invalidPCookieUserid.jsp`	Informs the user that a persistent cookie user name does not exist in the persistent cookie domain.
`invalidPassword.jsp`	Informs the user that the password entered does not contain enough characters.
`invalid_domain.jsp`	Informs the user that there is no such domain.
`Login.jsp`	This is a Login/Password template.
`login_denied.jsp`	Informs the user that no profile has been found in this domain.
`login_failed_template.jsp`	Informs the user that authentication has failed.
`Logout.jsp`	Informs the user that they have logged out.
`maxSessions.jsp`	Informs the user that the maximum sessions have been reached.
`membership.jsp`	A login page for the Self-registration module.
`Message.jsp`	A generic message template for a general error not defined in one of the other error message pages.
`missingReqField.jsp`	Informs the user that a required field has not been completed.
`module_denied.jsp`	Informs the user that the user does not have access to the module.
`module_template.jsp`	A customizable module page.
`new_org.jsp`	This page is displayed when a user with a valid session in one organization wants to login to another organization.
`noConfig.jsp`	Informs the user that no module configuration has been defined.

Table 5-2 List of Customizable JSP Templates *(Continued)*

File Name	Purpose
`noConfirmation.jsp`	Informs the user that the password confirmation field has not been entered.
`noPassword.jsp`	Informs the user that no password has been entered.
`noUserName.jsp`	Informs the user that no user name has been entered. It links back to the login page.
`noUserProfile.jsp`	Informs the user that no profile has been found. It gives them the option to try again or select New User and links back to the login page.
`org_inactive.jsp`	Informs the user that the organization they are attempting to authenticate to is no longer active.
`passwordMismatch.jsp`	This page is called when the password and confirming password do not match.
`profileException.jsp`	Informs the user that an error has occurred while storing the user profile.
`Redirect.jsp`	This page carries a link to a page that has been moved.
`register.jsp`	A user self-registration page.
`session_timeout.jsp`	Informs the user that their current login session has timed out.
`userDenied.jsp`	Informs the user that they do not possess the necessary role (for role-based authentication.)
`userExists.jsp`	This page is called if a new user is registering with a user name that already exists.
`userPasswordSame.jsp`	Called if a new user is registering with a user name field and password field have the same value.
`user_inactive.jsp`	Informs the user that they are not active.
`wrongPassword.jsp`	Informs the user that the password entered is invalid.

XML Files

XML files describe the authentication module-specific properties based on the Authentication Module Properties DTD. Access Manager defines an authentication module configuration file for each of the default authentication modules. By default, Authentication XML files are installed in the following directory: *IdentityServer_base*`/SUNWam/web-src/services/config/auth/default`. Table 5-3 provides descriptions of the authentication module configuration files.

Note that XML files are looked up from the deployed location. In previous Access Manager versions, the XML files were looked up from the installed location.

Table 5-3 List of Authentication Module Configuration Files

File Name	Purpose
`AD.xml`	Defines a Login screen for use with Active Directory authentication.
`Anonymous.xml`	For anonymous authentication, although there are no specific credentials required to authenticate.
`Application.xml`	Needed for application authentication.
`Cert.xml`	For certificate-based authentication although there are no specific credentials required to authenticate.
`HTTPBasic.xml`	Defines one screen with a header only as credentials are requested via the user's web browser.
`JDBC.xml`	Defines a Login screen for use with Java Database Connectivity (JDBC) authentication.
`LDAP.xml`	Defines a Login screen, a Change Password screen and two error message screens (Reset Password and User Inactive).
`Membership.xml`	Default data interface which can be used to customize for any domain.
`MSISDN.xml`	Defines a Login screen for use with Mobile Subscriber ISDN (MSISDN).
`NT.xml`	Defines a Login screen.
`RADIUS.xml`	Defines a Login screen and a RADIUS Password Challenge screen.
`SafeWord.xml`	Defines two Login screens: one for User Name and the next for Password.
`SAML.xml`	Defines a Logins screen for Security Assertion Markup Language (SAML) authentication.
`SecurID.xml`	Defines five Login screens including UserID and Passcode, PIN mode, and Token Passcode.
`Unix.xml`	Defines a Login screen and an Expired Password screen.

This following sections describe XML elements you can modify to customize the authentication UI. For a comprehensive list of authentication elements defined in the Authentication Module Properties DTD, see the *Developer's Reference*.

Callbacks Element

The *Callbacks* element is used to define the information a module needs to gather from the client requesting authentication. Each *Callbacks* element signifies a separate screen that can be called during the authentication process.

Nested Elements

The following table describes nested elements for the Callbacks element.

Element	Required	Description
`NameCallback`	*	Requests data from the user; for example, a user identification.
`PasswordCallback`	*	Requests password data to be entered by the user.
`ChoiceCallback`	*	Used when the application user must choose from multiple values.
`ConfirmationCallback`	*	Sends button information such as text which needs to be rendered on the module's screen to the authentication interface.
`HttpCallback`	*	

Attributes

The following table describes attributes for the Callbacks element.

Attribute	Default	Description
`length`		The number or length of callbacks
`order`		Is the sequence of the group of callbacks
`timeout`	60	Number of seconds the user has to enter credentials before the application times out.
`template`		Defines the UI or page level attributes for the UI customization
`image`		Defines the UI or page level attributes for the UI customization
`header`	Authentication	the text header information to be displayed on the UI
`error`	false	Indicates whether authentication framework/module needs to terminate the authentication process. If yes, then the value is true.

ConfirmationCallback Element

This element is used by the authentication module to send button information for multiple buttons. An example is the button text which needs to be rendered on the UI page. The element also receives the selected button information from the UI.

Nested Elements

The following table describes nested elements for the `ConfirmationCallback` element.

Element	Required	Description
OptionValues		

Attributes

None

Details

If there is only one button on the UI page then module is not required to send this callback.If Confirmation Callback is not provided through the Authentication Module properties XML file, then the global UI i18n properties file for all modules (`anAuthUI.properties`) will be used to pick and display the button text (label) for Login button.

Callbacks length value should be adjusted accordingly after addition of the new callback.

Example:

```
<ConfirmationCallback>
    <OptionValues>
        <OptionValue>
            <Value> <required button text> </Value>
        </OptionValue>
    </OptionValues>
</ConfirmationCallback>
```

JavaScript Files

JavaScript files are parsed within the `Login.jsp` file. You can add custom functions to the JavaScript files in the following directory: *IdentityServer_base*`/SUNWam/web-src/services/js`.

The JavaScript files used by the Authentication Service are summarized in Table 5-4.

Table 5-4 List of JavaScript Files

File Name	Purpose
`auth.js`	Used by `Login.jsp` for parsing all module files to display login requirement screens.
`browserVersion.js`	Used by `Login.jsp` to detect the client type.

Cascading Style Sheets

Modify the cascading style sheets (CSS) files to define the look and feel of the UI. Characteristics such as fonts and font weights, background colors, and link colors are specified in the CSS files. You must choose the appropriate .css file for your browser in order to customize the look and feel on the User Interface.

In the appropriate .css file, change the background-color attribute. Examples:

```
.button-content-enabled { background-color: red; }

button-link:link, a.button-link:visited { color: #000;
  background-color: red; text-decoration: none; }
```

There are a number of browser-based CSS files installed with Access Manager in the following directory:

IdentityServer_base`/SUNWam/web-src/ services/css`.

Table 5-5 provides a brief description of each CSS file.

Table 5-5 List of Cascading Style Sheets

File Name	Purpose
`css_generic.css`	Configured for generic web browsers.
`css_ie5win.css`	Configured specifically for Microsoft® Internet Explorer v.5 for Windows®.

Table 5-5 List of Cascading Style Sheets *(Continued)*

File Name	Purpose
`css_ns4sol.css`	Configured specifically for Netscape™ Communicator v. 4 for Solaris™.
`css_ns4win.css`	Configured specifically for Netscape Communicator v.4 for Windows.
`styles.css`	Used in JSP pages as a default style sheet.

Images

The default authentication GUI is branded with Sun Microsystems, Inc. logos and images. By default, the GIF files are installed in the following directory:

`SUNWam/web-src/services/login_images`

These images can be replaced with images relevant to your company. Table 5-6 provides a brief description for each GIF image used for the default GUI.

Table 5-6 List of Sun Microsystems Branded GIF Images

File Name	Purpose
`Identity_LogIn.gif`	Sun Java System Access Manager banner across the top.
`Registry_Login.gif`	No longer used.
`bannerTxt_registryServer.gif`	No longer used.
`logo_sun.gif`	Sun Microsystems logo in the upper right corner.
`spacer.gif`	A one pixel clear image used for layout purposes.
sunOne.gif	Sun Java System logo in the lower right corner.

Localization Files

Location: <install-dir>/SUNWam/locale

These are "i18n" properties files global to the Access Manager instance. A localization properties file, also referred to as an *i18n (internationalization) properties file* specifies the screen text and error messages that an administrator or user will see when directed to an authentication module's attribute configuration page. Each authentication module has its own properties file that follows the naming format amAuth*modulename*.properties; for example, amAuthLDAP.properties. They are located in *IdentityServer_base*/SUNWam/locale/. The default character set is ISO-8859-1 so all values are in English, but Java applications can be adapted to various languages without code changes by translating the values in the localization properties file.

Table 5-7 contains a listing of the localization properties files configured for each module. These files can be found in *IdentityServer_base*/SUNWam/locale.

Table 5-7 List of Localization Properties Files

File Name	Purpose
amAuth.properties	Defines the parent Core Authentication Service.
amAuthAD.properties	Defines the Active Directory Authentication Module.
amAuthAnonymous.properties	Defines the Anonymous Authentication Module.
amAuthApplication.properties	For Access Manager internal use only. Do not remove or modify this file.
amAuthCert.properties	Defines the Certificate Authentication Module.
amAuthConfig.properties	Defines the Authentication Configuration Module.
amAuthContext.properties	Defines the localized error messages for the AuthContext Java class.
amAuthContextLocal.properties	For Access Manager internal use only. Do not remove or modify this file.
amAuthHTTPBasic.properties	Defines the HTTP Basic Authentication Module.
amAuthJDBC.properties	Defines the Java Database Connectivity (JDBC) Authentication Module.
amAuthLDAP.properties	Defines the LDAP Authentication Module.
amAuthMembership.properties	Defines the Membership Authentication Module.
amAuthMSISDN.properties	Defines the Mobile Subscriber ISDN Authentication Module.
amAuthNT.properties	Defines the Windows NT Authentication Module.
amAuthRadius.properties	Defines the RADIUS Authentication Module.
amAuthSafeWord.properties	Defines the Safeword Authentication Module.
amAuthSAML.properties	Defines the Security Assertion Markup Language (SAML) Authentication Module.
amAuthSecurID.properties	Defines the SecurID Authentication Module.

Table 5-7 List of Localization Properties Files

File Name	Purpose
`amAuthUI.properties`	Defines labels used in the authentication user interface.
`amAuthUnix.properties`	Defines the UNIX Authentication Module.

Customizing Branding and Functionality

You can modify JSP templates and module configuration properties files to reflect branding or functionality specified for any of the following:

- Organization of the request
- SubOrganization of the request.
- Locale of the request
- Client Path
- Client Type information of the request
- Service Name (serviceName)

To Modify Branding and Functionality

1. Go to the directory where default JSP templates are stored.

   cd *AccessManager-base*/SUNWam/web-src/services/config/auth

2. Create a new directory.

   Use the appropriate customized directory path based on the level of customization. Use the following forms:

   ```
   org_locale/orgPath/filePath
   org/orgPath/filePath
   default_locale/orgPath/filePath
   ```

`default/orgPath/filePath`

In these examples,

`orgPath` represents `subOrg1/subOrg2`

`filePath` represents `clientPath + serviceName`

`clientPath` represents `clientType/sub-clientType`

Note that Sub-org, Locale, Client Path, Service Name (which represents `orgPath` and `filePath`) are optional. Note also that the organization name you specify may match the organization attribute set in the Directory Server. For example, if the organization attribute value is SunMicrosystems, then the organization customized directory should also be SunMicrosystems. If no organization attribute exists, then the lowercase value of the organization name (sunmicrosystems) should be used.

For example, for the following attributes:

`org` = SunMicrosystems

`locale` = en

`subOrg` = solaris

`clientPath` = html/*customerName*/

`serviceName` = paycheck

customized directory paths would be:

`SunMicrosystems_en/solaris/html/`*ustomerName*`/paycheck`

`SunMicrosystems/solaris/html/`*ustomerName*`/paycheck`

`default_en/solaris/html/`*ustomerName*`/paycheck`

`default/solaris/html/`*ustomerName*`/paycheck`

3. Copy the default templates.

Copy all the JSP templates (`*.jsp`) and authentication module configuration properties xml files (`*.xml`) from the default directory:

AccessManager-base`/SUNWam/web-src/services/config/auth/default`

to the new directory:

AccessManager-base`/SUNWam/web-src/services/config/`
`auth/`*CustomizedDirectoryPath*

4. Customize the files in the new directory.

   The files in the new directory can be customized if necessary, but not this is not required. See "Customizing the Login Page" on page 104 and "Customizing JSP Templates" on page 104 for information on what you can modify.

5. Update and redeploy `services.war`.

   Once you've modified the authentication GUI files, in order to see the changes in the actual GUI, you must update and then redeploy `services.war`. See "Updating and Redeploying services.war" on page 117 in this chapter for instructions. See Appendix C, "WAR Files" on page 359 for general information on updating and redeploying Access Manager `.war` files.

6. Restart both Access Manager and the web container server.

Customizing the Self-Registration Page

You can customize the Self-registration page which is part of Membership authentication module. The default data and interface provided with the Membership authentication module is generic and can work with any domain.You can configure it to reflect custom data and information.You can add custom user profile data or fields to register or to create a new user.

To Modify the Self-Registration Page

1. Customize the `Membership.xml` file.

   By default, the first three data fields are required in the default Membership Module configuration:

   - User name
   - User Password
   - Confirm User Password

   You can specify which data is requested, which is required, and which is optional. Code Example 5-1 on page 116 illustrates how to add a telephone number as requested data.

   You can specify or add data which should be requested from a user as part of the User Profile. By default you can specify or add any attributes from the following objectClasses:

- top
- person
- organizationalPerson
- inetOrgPerson
- iplanet-am-user-service
- inetuser

Administrators can add their own user attributes to the User Profile.

2. Update and redeploy `services.war`.

   Once you've modified the authentication GUI files, in order to see the changes in the actual GUI, you must update and then redeploy `services.war`. See "Updating and Redeploying services.war" on page 117 in this chapter for instructions. See Appendix C, "WAR Files" on page 359 for general information on updating and redeploying Access Manager `.war` files.

3. Restart both Access Manager and the web container server.

Code Example 5-1 Adding a Telephone Number as Requested Data

```
<Callbacks length="9" order="16" timeout="300" header="Self Registration"
template="register.jsp" >

    <NameCallback isRequired="true" attribute="uid" >
    <Prompt> User Name: </Prompt>
    </NameCallback>

    <PasswordCallback echoPassword="false" isRequired="true"
attribute="userPassword" >
    <Prompt> Password: </Prompt>
    </PasswordCallback>

    <PasswordCallback echoPassword="false" isRequired="true" >
    <Prompt> Confirm Password: </Prompt>
    </PasswordCallback>

    <NameCallback isRequired="true" attribute="givenname" >
    <Prompt> First Name: </Prompt>
    </NameCallback>

    <NameCallback isRequired="true" attribute="sn" >
    <Prompt> Last Name: </Prompt>
    </NameCallback>

    <NameCallback isRequired="true" attribute="cn" >
    <Prompt> Full Name: </Prompt>
    </NameCallback>
```

Code Example 5-1 Adding a Telephone Number as Requested Data *(Continued)*

```
    <NameCallback attribute="mail" >
    <Prompt> Email Address: </Prompt>
    </NameCallback>

     <NameCallback isRequired="true"attribute="telphonenumber">
     <Prompt> Tel:</Prompt>
     </NameCallback>

    <ConfirmationCallback>

        <OptionValues>
        <OptionValue>
        <Value> Register </Value>
        </OptionValue>
        <OptionValue>
        <Value> Cancel </Value>
        </OptionValue>
        </OptionValues>

    </ConfirmationCallback>

</Callbacks>
```

Updating and Redeploying services.war

If Access Manager is installed on BEA WebLogic, IBM WebSphere, or Sun ONE Application Server, you must update and redeploy `services.war` before you can see any changes in the user interface. Once you've made changes to the authentication GUI files, regardless of the brand of web container you're using, it is a good practice to update and redeploy the `services.war` file. When you update and redeploy `services.war`, you overwrite the default GUI files with your changes, and the changed files are placed in their proper locations. The section "services.war File" on page 103 provides background information on this file.

To Update services.war

1. cd *IdentityServer_base*/`SUNWam`

   This is the directory in which the WARs are kept.

2. jar -uvf *WARfilename*.war <*path_to_modified_file*>

   The `-uvf` option replaces the old file with the newly modified file. For example:

   ```
   jar -uvf services.war newfile/index.html
   ```

   replaces the `index.html` file in console.war with the `index.html` file located in *IdentityServer_base*/`SUNWam/newfile`.

3. `rm newfile/index.html`

   Deletes the modified file.

To Redeploy services.war

The `services.war` will be in the following directory:

```
AccessManager-base/SUNWam
```

Depending upon the brand of web container you are using, execute one of the following commands.

On BEA WebLogic

```
java weblogic.deploy -url ServerURL -component {ServerDeployURI}:
    {WL61 Server} deploy WL61AdminPassword {ServerDeployURI}
```

```
{AccessManager-base}/{SUNWam}/services.war
```

In this example,

ServerURL uses the form *protocol*://*host*:*port*
Example: `http://abc.com:58080`

ServerDeployURI represents the server Universal Resource Identifier
Example: `amserver`

WL61 Server represents the Weblogic Server name
Example: *name*.com

On Sun ONE Application Server

asadmin deploy -u *IAS7Admin* -w IAS7AdminPassword -H *HostName* -p IAS7*AdminPort* --type web SECURE_FLAG --contextroot

ServerDeployURI --name amserver --instance IAS7Instance
{*AccessManager-base*}`/{SUNWam}/services.war`

On IBM WebSphere

See the deployment documentation that comes with the IBM WebSphere product. websphere:

```
http://www-3.ibm.com/software/webservers/studio/doc/v40/studioguide/
    en/html/sdsscenario1.html
```

Chapter 6

Using Authentication APIs and SPIs

This chapter provides information on using Sun Java™ System Access Manager 6 2005Q1 authentication programming interfaces to use and to extend the Authentication Service.

This chapter contains the following sections:

Overview of Authentication APIs and SPIs

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. This communication between the APIs and the Authentication Service occurs by sending XML messages over `HTTP(S)`. The `remote-auth.dtd` is the template used in formatting the XML request messages sent to Access Manager and for parsing the XML return messages received by the external application. You can access `remote-auth.dtd` in the directory *AccessManager-base*/`SUNWam/dtd`.

New authentication modules are added to Access Manager by using the `com.iplanet.authentication.spi` package. The 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.

- How the Authentication Java APIs Work
- How the Authentication C-APIs Work
- XML / HTTP Interface for Other Applications

- How the Authentication SPIs Work

NOTE If contacting the Authentication Service directly through its URL (`http://`*identity_server_host.domain_name:port/service_deploy_uri*`/authservice`) without the API, a detailed understanding of `remote-auth.dtd` will be needed for generating and interpreting the messages passed between the client and server. Sample response and return XML messages can be found in "Examples of XML Messages" on page 124.

How the Authentication Java APIs Work

External Java applications can authenticate users with the Access Manager Authentication Service by using the Authentication Java APIs. The APIs are organized in a package called `com.sun.identity.authentication` and can be executed locally or remotely. The classes and methods defined in this package are used to initiate the authentication process and communicate authentication credentials to the specific modules within the Authentication Service. The classes and methods can be incorporated into a Java application to allow communication with the Authentication Service.

The first step necessary for an external Java application to authenticate to Access Manager is to create a new `AuthContext` object (`com.sun.identity.authentication.AuthContext`). The `AuthContext` class is defined for each authentication request as it initiates the authentication process. Since Access Manager can handle multiple organizations, `AuthContext` is initialized, at the least, with the name of the organization to which the requestor is authenticating. Once an `AuthContext` object has been created, the `login()` method is called indicating to the server what method of authentication is desired.

`IndexName` is the value of the authentication type. Table 6-1 `summarizes IndexName values and their corresponding authetnication types.`

Table 6-1 IndexName Values

IndexName Value	Authentication Type
`AuthContext.IndexType.ROLE`	Role-based
`AuthContext.IndexType.SERVICE`	Service-based
`AuthContext.IndexType.USER`	User-based
`AuthContext.IndexType.LEVEL`	Authentication Level-based
`AuthContext.IndexType.MODULE_INSTANCE`	Module-based

The `getRequirements()` method then calls the objects that will be populated by the user. Depending on the parameters passed with the instantiated `AuthContext` object and the two method calls, Access Manager responds to the client request with the correct login requirement screens. For example, if the requested user is authenticating to an organization configured for LDAP authentication only, the server will respond with the LDAP login requirement screen to supply a user name and a password. The client must then loop by calling the `hasMoreRequirements()` method until the required credentials have been entered. Once entered, the credentials are submitted back to the server with the method call `submitRequirements()`. The final step is for the client to make a `getStatus()` method call to determine if the authentication was successful. If successful, the caller obtains a session token for the user; if not, a `LoginException` is thrown.

Because the Authentication Service is built on the JAAS framework, the Authentication API can also invoke any authentication modules written purely with the JAAS API.

For detailed information about Java APIs for authentication, see the Javadoc in the following directory:

AccessManager-base`/SUNWam/docs`

How the Authentication C-APIs Work

C applications can authenticate users with the Access Manager Authentication Service by using the Authentication C-API. The C application contacts the Authentication Service to initiate the authentication process, and the Authentication Service responds with a set of requirements. The client application submits authentication credentials back to the Authentication Service and receives further authentication requirements back until there are no more to fulfill. After all requirements have been sent, the client makes one final call to determine if authentication has been successful or has failed. The C API can be found in *IdentityServer_base*`/SUNWam/agents`. This directory also includes a C API `samples` directory.

CAUTION Previous releases of Access Manager contained C libraries in *IdentityServer_base*`/lib/capi`. The `capi` directory is being deprecated, and is currently available for backward compatibility. It will be removed in the next release, and therefore it is highly recommended that existing application paths to this directory are changed and new applications do not access it. Paths include `RPATH`, `LD_LIBRARY_PATH`, `PATH`, compiler options, etc.)

The sequence of calls necessary to authenticate to Access Manager begins with the function call `am_auth_create_auth_context`. This call will return an `AuthContext` structure used for the rest of the authentication calls. Once an `AuthContext` structure has been initialized, the `am_auth_login` function is called. This indicates to the Authentication Service that an authentication is desired. Depending on the parameters passed when creating the `AuthContext` structure and making the `am_auth_login` function call, the Authentication Service will determine the login requirements with which to respond. For example, if the requested authentication is to an organization configured for LDAP authentication, and no authentication module chaining is involved, the server will respond with the requirements to supply a user name and a password. These attributes correspond to elements in the remote-auth.dtd structure. The user name corresponds to the NameCallback element; the password which corresponds to the PasswordCallback element. The client loops on function call `am_auth_has_more_requirements` (in this specific case there will be two). The client then fills in the needed information and submits this back to the server with function call am_auth_submit_requirements. The final step is to make function call `am_auth_get_status` to determine if the authentication was successful or not.

XML/HTTP Interface for Other Applications

Applications written in a programming language other than Java or C can exchange authentication information with Access Manager using the XML / HTTP(s) interface. Using the URL `http://`*server_name.domain_name*`:`*port*`/`*service_deploy_uri*`/authservice`, an application can open a connection using the HTTP POST method and exchange XML messages with the Authentication Service. The structure of the XML messages is defined in `remote-auth.dtd`. In order to access the Authentication Service in this manner, the client application must contain the following:

- A means of producing valid XML compliant with the `remote-auth.dtd`.
- HTTP 1.1 compliant client implementation to send XML-configured information to Access Manager.
- HTTP 1.1 compliant server implementation to receive XML-configured information from Access Manager.
- An XML parser to interpret the data received from Access Manager.

Examples of XML Messages

The following code examples illustrate how customers might configure the XML messages posted to the Authentication Service.

NOTE Although the client application need only write XML based on the `remote-auth.dtd`, when these messages are sent they include additional XML code produced by the Authentication API. This additional XML code is not illustrated in the following examples.

Code Example 6-1 illustrates the initial XML message sent to the Access Manager. It opens a connection and asks for authentication requirements regarding the `exampleorg` organization to which the user will login.

Code Example 6-1 Initial AuthContext XML Message

```
<?xml version="1.0" encoding="UTF-8"?>
<AuthContext version="1.0">
<Request authIdentifier="0">
<NewAuthContext orgName="dc=exampleorg,dc=com">
</NewAuthContext>
</Request>
</AuthContext>
```

Code Example 6-2 illustrates the successful response from Access Manager that contains the `authIdentifier`, the session identifier for the initial request.

Code Example 6-2 AuthIdentifier XML Message Response

```
<?xml version="1.0" encoding="UTF-8"?>
<AuthContext version="1.0">
<Response
authIdentifier="AQIC5wM2LY4SfcwmVdbgTX+9WzyWSPlWjb1oVb5esqDlkaY=">
<LoginStatus status="in_progress">
</LoginStatus>
</Response>
</AuthContext>
```

Code Example 6-3 illustrates the client response message back to Access Manager. It specifies the type of authentication module needed by the user to log in.

Code Example 6-3 Second Request Message With Authentication Module Specified

```
<?xml version="1.0" encoding="UTF-8"?>
<AuthContext version="1.0">
<Request authIdentifier="AQIC5wM2LY4SfcwmVdbgTX+9WzyWSPlWjb1oVb5esqDlkaY=">
<Login>
<IndexTypeNamePair indexType="moduleInstance">
<IndexName>LDAP</IndexName>
</IndexTypeNamePair>
</Login>
</Request>
</AuthContext>
```

Code Example 6-4 illustrates the return message from Access Manager which specifies the authentication module's login requirements. In this case, the LDAP requirements include a user name and password. Note the page time out value of 120 seconds.

Code Example 6-4 Return XML Message With Login Callbacks

```
<?xml version="1.0" encoding="UTF-8"?>
<AuthContext version="1.0">
<Response
authIdentifier="AQIC5wM2LY4SfcwmVdbgTX+9WzyWSPlWjb1oVb5esqDlkaY=">

<GetRequirements>
<Callbacks length="3">
<PagePropertiesCallback isErrorState="false">
<ModuleName>LDAP</ModuleName>
<HeaderValue>This server uses LDAP Authentication</HeaderValue>
<ImageName></ImageName>
<PageTimeOut>120</PageTimeOut>
<TemplateName></TemplateName>
<PageState>1</PageState>
</PagePropertiesCallback>
<NameCallback>
<Prompt>User Name: </Prompt>
</NameCallback>

<PasswordCallback echoPassword="false">
<Prompt> Password: </Prompt>
</PasswordCallback>

</Callbacks>
</GetRequirements>
</Response>
</AuthContext>
```

Code Example 6-5 illustrates the client responses to the call for login requirements. They specify `amadmin` as the user and `11111111` for the password.

Code Example 6-5 Response Message With Callback Values

```
<?xml version="1.0" encoding="UTF-8"?>
<AuthContext version="1.0">
<Request authIdentifier="AQIC5wM2LY4SfcwmVdbgTX+9WzyWSPlWjb1oVb5esqDlkaY=">
<SubmitRequirements>
<Callbacks length="3">

<NameCallback>
<Prompt>User Name:</Prompt>
<Value>amadmin</Value>
</NameCallback>

<PasswordCallback echoPassword="false">
<Prompt>Password:</Prompt>
<Value>11111111</Value>
</PasswordCallback>
</Callbacks>
</SubmitRequirements>
</Request>
</AuthContext>
```

Code Example 6-6 illustrates that a successful authentication has occurred. As the value of `<Subject>` uses the Java serialization, it can not be used by non-Java client applications. It's value is retrieved by all applications from the session token.

Code Example 6-6 Successful Authentication XML Message

```
<?xml version="1.0" encoding="UTF-8"?>
<AuthContext version="1.0">
<Response
authIdentifier="AQIC5wM2LY4SfcwmVdbgTX+9WzyWSPlWjb1oVb5esqDlkaY=">
<LoginStatus status="success"
ssoToken="AQIC5wM2LY4SfcwmVdbgTX+9WzyWSPlWjb1oVb5esqDlkaY="
successURL="http://torpedo.red.iplanet.com:/amconsole">
```

Code Example 6-6 Successful Authentication XML Message

```
<Subject>AQICweczOhuelZ5TqD9kKOtiAepxqGP23q4oTnNMuJY//lI2S4KD1/gEN84uLwDGHl
llyFSthxoKLM7NDH
h2vwAvrDmpsomJvUnbqnJJ90DS+28njGiDv+lv8FqIVhhbxrctbiIUEOHYK0FzXnXjPYizdCmiW
XJ+9DJ8T2HbYIDxn9U6eVNAMPq3uVb/RFuErEm5MuPu7PnWeCic12SZre4ZEcw8TI45NKNjd/NZ
ZD97bcqL5gEV7SVHspFldZKmo9vA86aEkvMs9P53RiJtrusHN1FKt9+4JqSrdcVLKMzJVAr3z5E
ohwHh9/hzd7hguc0661gz7IqkT7WEpve/E8R4em0mg3HgHg7Bg7i3AkyX6YSkoAncdVXMdmWnb7
OV5cBgUjO8zs8Pp5/3dA1XlwACmOqjxshk6Y6Ld6TAQ90qRFwymC1RdLGGCRnrt33kmYVyB1lJy
JxT8utPKyDOEKFRHh57NlKTbFhBKc1IGcdQ2crHifpXawx6YouQgQSWGdsqW9IahY4+lqbBTPnG
DyZkKz9yy2ZKVjDR05Hwku8elvEwBE40XTJ3gF/mbwCGbh3cyprahLqRXboy8eoEQf3ubQmR2My
+bh+NrsRfzfFV5oCcpJE6DtvYE/4zO+uKk3FbG+/NUJzAAor920V/0prtYeS58ZPW8C7qwXINaW
0xdMQV+pgE3NZvMlp5GeZlSIMmSCtXD49n4tqopSlsoK+eiwPODKxp992+6/uJhhVHH5I0Ozuy6
CDM
dCJDGvnMENVCUZvki3+tb92fqQbVWixM4Ca6Nnz3jTIKk2uhm559jq9hra8gHHOfnnu4e5jZjzf
RdkO3GodiTMOHDnQATHtvT1PBXgorTfUwUa4ZjptvzFulHSi4eQaqs4Z8FAX2OAr8XGHRkhBwox
rhjYiCDBpkNmpEiFNhWnTT3bwkAUFhtoDg6836kwHfxeLXKAz3T6qyNQzT+larSXUxrt/TIjwDP
R3vg4GF4RzbHlWA1WQtUS/9Qe/N3aegEEEvxPvo9fWq</Subject>
</LoginStatus>
</Response>
</AuthContext>
```

How the Authentication SPIs Work

- Extending the AMLoginModule Class
- Pluggable JAAS Module
- Authentication Post Processing

Access Manager provides the capability to plug new, Java-based authentication modules into its framework allowing proprietary authentication providers to be managed using the Access Manager console. A custom authentication module must first be created using Java. Once created, the custom module can be added to the list of available authentication modules.

NOTE This guide does not document the JAAS. For more information on these APIs, see the *Java Authentication And Authorization Service Developer's Guide*. Additional information can be found at `http://java.sun.com/products/jaas/`.

Extending the AMLoginModule Class

Custom authentication modules extend the `com.sun.identity.authentication.spi.AMLoginModule` class. The class must also implement the `init()`, `process()` and `getPrincipal()` methods in order to communicate with the authentication module configuration files. The callbacks are then dynamically generated based on this file. Other methods that can be defined include `setLoginFailureURL` and `setLoginSuccessURL` which defines URLs to send the user to based on a failed or successful authentication, respectively.

NOTE To make use of the account locking feature with custom authentication modules, the `InvalidPasswordException` exception should be thrown when the password is invalid.

Pluggable JAAS Module

The Java Authentication and Authorization Service (JAAS) is a set of APIs that enable services to authenticate and enforce access controls upon users. It implements a Java technology version of the standard Pluggable Authentication Module (PAM) framework, and supports user-based authorization. Access Manager supports pure JAAS pluggable authentication modules. In Access Manager, pure JAAS modules extend the JAAS `LoginModule` rather than `AMLoginModule`. A pure JAAS module is plugged in to the Authentication framework using the Authentication API.

Authentication Post Processing

The Authentication SPI includes the `AMPostAuthProcessInterface` which can be implemented for post-processing tasks. The following are examples of post-processing tasks:

- Adding attributes to a user’s session after successful authentication
- Sending notification to an administrator after failed authentication
- General clean-up such as clearing cookies after logout or logging out of other system components.

The Core Authentication Service contains the Authentication PostProcessing Class attribute which contains the authentication post-processing class name as its value. Custom post processing interfaces can also be implemented.

`AMPostAuthProcessInterface` can be implemented for post authentication processing on authentication success, failure and logout. The SPI is configurable at the organization , service and role levels. The Authentication Service invokes the post processing SPI methods on successful, failed authentication and logout.

The `AMPostProcessInterface` class has 3 methods:

- onLoginSuccess
- onLoginFailure
- onLogout

Some supporting information on these methods is provided in the following sections. For a comprehensive listing and detailed information on all Access Manager methods, see the Javadoc installed in the following directory:

`AccessManager-base/SUNWam/docs`

onLoginSuccess

This method should be implemented for post-processing after a successful authentication. Authentication Service will invoke this method on successful authentication.

Method signature is:

```
public void onLoginSuccess(Map requestParamsMap,
                              HttpServletRequest request,
                              HttpServletResponse response,
                              SSOToken ssoToken)
    throws AuthenticationException;
```

where

- `requestMap` is a map containing `HttpServletRequest` parameters
- request `HttpServletRequest object`
- `response HttpServletResponse object`

`com.sun.identity.authentication.spi.AuthenticationException` is thrown on error.

onLoginFailure

This method should be implemented for post processing after a failed authentication. Authentication Service will invoke this method on failed authentication.

Method signature is:

```
public void onLoginFailure(Map requestParamsMap,
                            HttpServletRequest request,
                            HttpServletResponse response)
   throws AuthenticationException;
```

where

- `requestMap` is a map containing `HttpServletRequest` parameters
- request `HttpServletRequest` object
- response `HttpServletRequest` object

`com.sun.identity.authentication.spi.AuthenticationException` is thrown on error.

onLogout

This method should be implemented for post-processing on a logout request. Authentication Service will invoke this method on logout.

Method signature is:

```
public void onLogout(HttpServletRequest request,
                      HttpServletResponse response,
                     SSOToken ssoToken)
   throws AuthenticationException;
```

where

- request HttpServletRequest object is a map containing `HttpServletRequest` parameters
- response `HttpServletResponse` object
- ssoToken authenticated user's single sign on token

`com.sun.identity.authentication.spi AuthenticationException` is thrown on error.

Using Authentication APIs

Access Manager comes with a number of sample programs that demonstrate how you can use the Authentication APIs to extend the functionality of the authentication service and authentication modules.

- Running the Sample Authentication Programs
- LDAPLogin Example
- CertLogin Example
- JCDI Module Example
- C-API Sample

Running the Sample Authentication Programs

The source code and Makefile are provided for all sample programs. For some sample programs, additional supporting files are also included.The instructions for compiling and executing the sample programs are the same for all samples described in this section. See "To Compile and Execute the Java API Samples." Instructions for configuring the sample programs to run in Secure Sockets Layer (SSL) mode are also included in this section. See "To Configure SSL for Java API Samples."

Java API Code Samples and Their Locations

The following tables describe the locations of all the files you need to implement the sample programs on various platforms, and the variable names used for default directories in the source code and Makefiles. Table 6-2 summarizes file locations and variable names used for Solaris Sparc/x86.l Table 6-3 summarizes default directories for Linux. Table 6-4 summarizes default directories for Windows 2000.

Table 6-2 Default directories for Solaris Sparc/x86

Variable	Description	Location
Api_sample_dir	Directory that contains authentication API sample files	`<install_root>/SUNWam/samples/authenitcation/api`
Config_directory	Directory that contains configuration files	`/etc/opt/SUNWam/config`
Product_Directory	Directory where Access Manager is installed.	`install_root>/SUNWam`

Table 6-3 Default directories for Linux

Variable	Description	Location
Api_Sample_Dir	Directory that contains authentication API sample files	`<install_root>/sun/identity/samples/authentication/api`
Config_Directory	Directory that contains configuration files	`/etc/opt/sun/identity/config`
Product_Directory	Directory where Access Manager is installed.	`<install_root>/sun/identity`

Table 6-4 Default directories for Windows 2000

Variable	Description	Location
Api_Sample_Dir	Directory that contains authentication API sample files	`<install_root>\samples\authentication\api`
Config_Directory	Directory that contains configuration files	`<install_root>\lib`
Product_Directory	Directory where Access Manager is installed.	`<install_root>`

To Compile and Execute the Java API Samples

These steps are for all platforms.

1. In the Makefile, modify the following variables as necessary to suit your Access Manager installation:

   BASE_DIR: Enter the path to the directory where Access Manager is installed.

   JAVA_HOME: Enter the path to the directory where the Java compiler is installed.

   DOMAIN: Enter the name of the organization to login to.

   SHARE_LIB: Enter the path to the directory where Access Manager jar files are stored.

   JSS_JAR_PATH: Enter the path to the directory where JSS jar files are stored.

   JSSPATH: Enter the path to the directory where JSS libraries are located.

2. In the Certificate Sample Makefile only, modify the following as necessary:

   CERTNICKNAME: Enter the Certificate nickname.

   URL: Enter the Access Manger Server URL.

   PASSWORD: Enter the Certificate DB Password.

3. Copy `AMConfig.properties` from *Config_Directory* in the Access Manager server installation to the client machine.

   (Note: For SSL check SSL Configuration Setup, step 2).

4. In the Makefile, update the classpath to include the location of the newly created `AMConfig.properties`.

5. In the client machine, create a directory named `locale`. Then copy all the property files from the `locale` directory in the Access Manager server installation machine to the client machine. The locale directory on the server machine can be found under the *Product_Directory*.

6. Update the classpath in the Makefile to include the location of newly created `locale` files.

7. Include `jaas.jar` in your classpath if you are using a JDK version less than JDK1.4

8. Compile the program.

   - On Solaris Sparc/x86, Linux, run the `gmake` command.
   - On Windows 2000, run the `make` command.

9. Run the sample program.

   - On Solaris Sparc/x86 or Linux, run the following command: `gmake run`
   - On Windows 2000, run the following command: `make run`

To Configure SSL for Java API Samples

1. In the Makefile, add this JVM property in the `run` target:

   ```
   -D "java.protocol.handler.pkgs=com.iplanet.services.comm"
   ```

2. Copy `AMConfig.properties` from *Config_Directory* in the Access Manager server installation to the client machine.

3. Edit the following properties in `AMConfig.properties`.

   com.iplanet.am.admin.cli.certdb.dir: Enter the path to the certificate database directory.

   com.iplanet.am.admin.cli.certdb.prefix: Enter the certificate database prefix.

4. In the LDAP and JCDI Samples only:

   com.iplanet.am.server.protocol: Change the value to https.

   com.iplanet.am.server.port: Enter the appropriate port number from the server machine.

5. Create or copy the certificate database file to the certificate db directory. Use the directory name in `com.iplanet.am.admin.cli.certdb.dir`.

6. Rename the file to use the prefix specified in the property `com.iplanet.am.admin.cli.certdb.prefix`.

For the details, see the Javadoc for the Remote Client API.

LDAPLogin Example

The LDAPLogin sample is an example of a custom Java application that uses the authentication remote APIs to authenticate to the LDAP module. You can modify the sample source code to authenticate to other existing or customized authentication modules. The sample source code, Makefile, and Readme.html are located in the following directory:

IdentityServer_base`/SUNWam/samples/authentication/LDAP`

To compile and run the sample program, follow the steps in "To Compile and Execute the Java API Samples" on page 133.

CertLogin Example

The CertLogin sample is an example of a custom Java application that uses digital certificates for authentication. You can modify the sample source code to authenticate to other existing or customized authentication modules. The sample source code, `Makefile`, and `Readme.html` are located in the following file:

IdentityServer_base`/SUNWam/samples/authentication/Cert`

To Run the CertLogin Program

1. Enable SSL.

   Follow the instructions in "To Configure SSL for Java API Samples" on page 134.

2. Compile and execute the sample code.
   "To Compile and Execute the Java API Samples" on page 133.

Using certutil for Client Certificate Management

`Certutil` is a command-line utility that can create and modify cert7.db and key3.db database files. It can also list, generate, modify, or delete certificates within the cert7.db file and create or change the password, generate new public and private key pairs, display the contents of the key database, or delete key pairs within the key3.db file. The key and certificate management process usually begins with creating keys in the key database, then generating and managing certificates in the certificate database.

JCDI Module Example

The JCDI Module Example demonstrates the use of Java Card Digital ID (JCDI) authentication with Access Manager. The sample has two components:

- Remote client
- Server JCDI authentication module

The remote client component is located in the following directory:

Identity_Server_base/`samples/authentication/api/jcdi`

The server JCDI authentication module is located in the following directory:

Identity_Server_base/`samples/authentication/spi/jcdi`

The sample illustrates JCDI authentication using the Remote Authentication API.You can modify the sample source code to authenticate to other existing or customized authentication modules. The source code, `Makefile`, and `Readme.html` are located in the following directory:

Identity_Server_base/`samples/authentication/api/jcdi`

To compile and run the sample program, follow the steps in "To Compile and Execute the Java API Samples" on page 133.

C-API Sample

Access Manager provides a sample program to demonstrate how an external C application can use the API to authenticate a user via Access Manager. The sample can be found in *IdentityServer_base*/`SUNWam/agents/samples/common/`.

By default, the C Authentication sample checks the directory *IdentityServer_base*/`SUNWam/agents/config` for a properties file named `AMAgent.properties`.

C Authentication Sample Properties. Code Example 6-7 lists the properties that are needed by the C Authentication API. Some of these are defined in `AMAgent.properties` and some are not. Those that are not defined in `AMAgent.properties` can be added to the file so they needn't be identified for each function call. For example, `com.sun.am.auth.orgName`, which identifies the organization from which you want to authenticate, can be added to `AMAgent.properties`.

C Header File. The C Authentication API header file, `am_auth.h`, can be found in *IdentityServer_base*/`SUNWam/agents/include`. It contains the function prototypes for the function calls available in the C Authentication API.

Code Example 6-7 AMAgent.properties File

```
# SOME PROPERTIES LISTED ARE NOT PRE-EXISTING IN THE PROPERTIES FILE

# the identity server naming service url
com.sun.am.namingURL=http://serverexample.domain.com:58080/amserver/namings
ervice
# the directory to use for logging
com.sun.am.logFile=/home/uid/logs/auth-log
# the logging level, all:5 being the highest and all:3 being medium
com.sun.am.logLevels=all:5
# the directory containing the certificate and key databases
com.sun.am.sslCertDir=/home/level/certdir
# the prefix of the cert7.db and key3.db files, if any
com.sun.am.certDbPrefix=
# the password to the key3.db file
com.sun.am.certDBPassword=11111111
# true to trust SSL certificates not in the client cert7.db
com.sun.am.trustServerCerts=true
# the nick name of the client certificate in the cert7.db
com.sun.am.auth.certificateAlias=Cert-Nickname
# the identity server organization desired for authentication
com.sun.am.auth.orgName=dc=sun,dc=com
```

Using Authentication SPIs

Access Manager provides the following sample programs to demonstrate how you can use the Authentication service provider interfaces (SPIs) to extend authentication functionality:

- Implementing a Custom Authentication Module
- Implementing Authentication PostProcessing SPI
- Generating an Authentication User ID
- Implementing A Pure JAAS Module

Implementing a Custom Authentication Module

Access Manager contains a sample exercise for integrating a custom authentication module with files that have already been created. This sample illustrates the steps for integrating an authentication module into the Access Manager deployment. All the files needed to compile, deploy and run the sample authentication module can be found in the following directory:

IdentityServer_base`/SUNWam/samples/authentication/providers`

The following sections will use files from this sample as example code:

- Writing a Sample Login Module
- Compiling and Deploying the LoginModule program
- To Deploy the Login Module Sample Program
- Loading the Login Module Sample into Access Manager
- Running the LoginModule Sample Program
- Deploying the Login Module Sample Program

About the Login Module Sample

<PRODUCT_DIR> setting on different Platforms:

Solaris Sparc/x86: <PRODUCT_DIR> = *base-directory* / SUNWam

Linux: <PRODUCT_DIR> = *base-directory* / sun / identity

W2K: <PRODUCT_DIR> = *base-directory*

Writing a Sample Login Module

Use the `AMLoginModule` SPI (Service Provider Interface) to write your own sample login module. Three steps are required:

1. Create a Module Properties File.
2. Write the Principal Class.
3. Implement the LoginModule Interface.

The following are the default directories used in the sample exercise for the various platforms:

Solaris Sparc/x86: <PRODUCT_DIR> = *base-directory* / SUNWam

Linux: <PRODUCT_DIR> = *base-directory* / sun / identity

W2K: <PRODUCT_DIR> = *base-directory*

Create a Module Properties File

Create a Module properties XML file with the same name of the class (no package name) and use the extension `.xml`. You must create an XML file with this naming convention even if no states required

Based on this configuration file, the Authentication user interface will dynamically generate a login page.

You can define page states in the module properties file as shown in Code Example 6-8. Each callback element corresponds to one login page state. When an authentication process is invoked, `Callback[]` values will be generated from the user's Login Module for each state. All login state definitions start with 1. The module controls the login process, and then determines what the next state is.

`Auth_Module_Properties.dtd` defines the data structure that will be used by each authentication module to specify its properties. `Auth_Module_Properties.dtd` provides definitions to initiate, construct and send required callbacks information to the Authentication UI. `Auth_Module_Properties.dtd` is stored in the <PRODUCT_DIR>`/dtd` directory.

Code Example 6-8 Module Configuration Sample

```
<ModuleProperties moduleName="LoginModuleSample" version="1.0" >
        <Callbacks length="2" order="1" timeout="60" header="This is a
sample login page" >
            <NameCallback>
        <Prompt> User Name </Prompt>
    </NameCallback>
    <NameCallback>
        <Prompt> Last Name </Prompt>
    </NameCallback>
        </Callbacks>
        <Callbacks length="1" order="2" timeout="60" header="You made it to
page 2" >
        <PasswordCallback echoPassword="false" >
        <Prompt> Just enter any password </Prompt>
            </PasswordCallback>
        </Callbacks>
    </ModuleProperties>
```

In the module configuration sample in Code Example 6-8, `page state` one has two callbacks. The first callback is for user ID, and second is for Last Name. When the user fills in the callbacks, the following events occur:

1. The `Callback[]` values are sent to the module.
2. The `process()` routine validates the callback values, and then returns what?.
3. The module writer sets the next page state to 2.

Page state 2 has one callback to request the user to enter a password. The `process()` routine is again called after the user submits the `Callback[] values.` If the module writer throws a `LoginException`, then an Authentication Failed page will be sent to the user. If no exception is thrown, the user is redirected to his or her default page.

Write the Principal Class

After creating module configuration xml file, the next step is to write a Sample Principal class which implements `java.security.Principal`. The constructor takes the user's username as an argument. If authentication is successful, the module will return this principal to Authentication framework. The Authentication framework populates a Subject with a `SamplePrincipal` representing the user.

Implement the LoginModule Interface

`AMLoginModule` is an abstract class which implements `JAAS LoginModule`. `AMLoginModule` provides methods for accessing Access Manager services and the module XML configuration. `Login Module` writers must subclass `AMLoginModule` class and implement the following methods:

- `init()`
- `process()`
- `getPrincipal()`

For detailed descriptions, syntax, and parameters, see the Javadoc. The following provides some supporting information about these methods.

`init()` This is an abstract method, Module writer should implement to initialize this LoginModule with the relevant information. If this LoginModule does not understand any of the data stored in sharedState or options parameters, the data can be ignored. This method is called by a AMLoginModule after this SampleLoginModule has been instantiated, and prior to any calls to its other public methods. The method implementation should store away the provided arguments for future use. The init method may additionally peruse the provided sharedState to determine what additional authentication state it was provided by other LoginModules, and may also traverse through the provided options to determine what configuration options were specified to affect the LoginModule's behavior. It may save option values in variables for future use.

`process()` The process method is called to authenticate a Subject. This method implementation should perform the actual authentication. For example, it may cause prompting for a user name and password, and then attempt to verify the password against a password database. If your LoginModule requires some form of user interaction (retrieving a user name and password, for example), it should not do so directly. That is because there are various ways of communicating with a user, and it is desirable for LoginModules to remain independent of the different types of user interaction. Rather, the LoginModule's process method should invoke the handle method of the CallbackHandler passed to this method to perform the user interaction and set appropriate results, such as the user name and password and the AMLoginModule internally passes the UI an array of appropriate Callbacks, for example a NameCallback for the user name and a PasswordCallback for the password, and the UI performs the requested user interaction and sets appropriate values in the Callbacks.

Consider the following points while writing the `process()` method:

- Perform the authentication. If Authentication succeeded, save the principal who has successfully authenticated.
- Return -1 if authentication succeeds, or throw a LoginException such as AuthLoginException if authentication fails or return relevant state specified in module configuration XML file
- If multiple states are available to the user, the Callback array from a previous state may be retrieved by using the getCallbak(int state) methods. The underlying login module keeps the Callback[] from the previous states until the login process is completed.

- If a module writer needs to substitute dynamic text in next state, the writer could use the getCallback() method to get the Callback[] for the next state, modify the output text or prompt, then call replaceCallback() to update the Callback array. This allows a module writer to dynamically generate challenges, passwords or user IDs. Note: Each authentication session will create a new instance of your Login Module Java class. The reference to the class will be released once the authentication session has either succeeded or failed. It is important to note that any static data or reference to any static data in your Login module must be thread-safe.

`getPrincipal()` This method should be called once at the end of a successful authentication session. A login session is deemed successful when all pages in the Module properties XML file have been sent and the module has not thrown an exception. The method retrieves the authenticated token string that the authenticated user will be known by in the Access Manager environment.

Compiling and Deploying the LoginModule program

If you are writing your own Custom Authentication module based on the `AMLoginModule` SPI or a pure JAAS module, then you can skip this step. Otherwise, after writing the sample Login Modue, compile and deploy the sample found under *AccessManager-base*`/samples/authentication/spi/providers`.

To compile the Login Module

1. Set the following environment variables.

   These variables will be used to run the `gmake` command. You can also set these variables in the Makefile. This Makefile is in the following directory: *AccessManager-base*`/samples/authentication/spi/providers`.

   JAVA_HOME: Set this variable to your installation of JDK. The JDK should be version 1.3.1_06 or higher.

   CLASSPATH: Set this variable to refer to am_services.jar which can be found in the *Idetnity_base*/lib directory. (Note: Include `jaas.jar` in your classpath if you are using JDK version less than JDK1.4)

   BASE_DIR: Set this variable to the directory where the Access Manager is installed.

   BASE_CLASS_DIR: Set this variable to the directory where all the Sample compiled classes are located.

   JAR_DIR: Set this variable to the directory where the JAR files of the Sample compiled classes will be created.

2. In the *AccessManager-base*/`samples/authentication/spi/providers` directory, run `gmake`.

To Deploy the Login Module Sample Program

1. Copy `LoginModuleSample.jar` from *JAR_DIR* to *AccessManager-base*/`web-src/services/WEB-INF/lib`.

2. Copy LoginModuleSample.xml from *AccessManager-base*/`samples/authentication/spi/providers` to *AccessManager-base*/`web-src/services/config/auth/default`.

3. Redeploy the `amserver.war` file.

   See Appendix C, "WAR Files" on page 359 for detailed information on deploying Access Manager `.war` files.

Loading the Login Module Sample into Access Manager

Once you've compiled and deployed the login module, you must load the login module into Access Manager. You can load the login module by using either the Access Manager administration console, or by using the `amadmin` command.

To Load the Login Module Using the Administration Console

1. Login to Access Manager Console as amadmin, using the URL:
   `http://<host>.<domain>:<port>/<Console-Deploy-URL>`

2. Click Service Configuration.

3. In Service Configuration frame, under Authentication, click Core.

4. Add class file name
   ```
   com.iplanet.am.samples.authentication.spi.providers.
        LoginModuleSample
   ```
   to the Pluggable Auth Modules Classes list.

5. Click Save.

To Load the Login Module Using using the Command Line

1. Write a sample XML file as shown in Code Example 6-9, which will add the `LoginModuleSample` auth module entry into the allowed modules and an authenticators list.

Code Example 6-9 Adding the LoginModuleSample entry.

```
        <!--
            Copyright (c) 2003 Sun Microsystems, Inc. All rights reserved
```

Code Example 6-9 Adding the LoginModuleSample entry.

```
              Use is subject to license terms.
         -->

          <!DOCTYPE Requests
               PUBLIC "-//iPlanet//iDSAME 5.0 Admin CLI DTD//EN"
              "jar://com/iplanet/am/admin/cli/amAdmin.dtd"
          >

         <Requests>

             <SchemaRequests serviceName="iPlanetAMAuthService"
SchemaType="Global">
               <AddDefaultValues>
                <AttributeValuePair>
                   <Attribute name="iplanet-am-auth-authenticators"/>


<Value>com.iplanet.am.samples.authentication.spi.providers.LoginModuleSampl
e</Value>
                 </AttributeValuePair>
               </AddDefaultValues>

             </SchemaRequests>
         </Requests>
```

2. Use `amadmin` to load `sample.xml`:

```
<AMADMIN> --runasdn uid=amAdmin,ou=People,<root_suffix> --password
    <password> --data sample.xml
```

Solaris Sparc/x86: `AMADMIN = <PRODUCT_DIR>/bin/amadmin`

On W2K: `AMADMIN = <PRODUCT_DIR>\bin\amadmin`

Running the LoginModule Sample Program

This sections provides instructions for running the login module on Solaris and on Windows platforms.

To Run the LoginModule on Solaris

1. Use the following URL to log in to Access Manager console as amAdmin: `http://<host>.<domain>:<port>/<Console-Deploy-URI>`

2. Click Identity Management, and in the Identity Management view select your organization.

3. From the View menu, select Services.

4. In the navigation frame, under Authentication, click Core.
5. Select LoginModuleSample to add it to the list of highlighted modules in Organization Authentication Modules.

   Make sure LDAP module is also selected. If not selected, you will not be able to login to Access Manager Console. You can use Control + mouse click to add additional modules.
6. Click Save.
7. Log out.
8. Enter the following URL:

   ```
   http://<host>.<domain>:<port>/<Service-Deploy-URI>/UI/Login?module=
       LoginModuleSample
   ```

   If you choose to use an organization other than the default, be sure to specify that in the URL using the `org` parameter.

To Run the Login Module on Windows 2000

1. Set the following environment variables. These variables will be used to run the make command. You can also set these variables in the Makefile.

   This Makefile is in the same directory as the Login Module Sample program files: *AccessManager-base*`\samples\authentication\spi\providers`

   JAVA_HOME: Set this variable to your installation of JDK. The JDK should be version 1.3.1_06 or higher.

   BASE: Set this variable to *base-directory*

   CLASSPATH: Set this variable to refer to am_services.jar which can be found in the *base-directory*\lib directory. (Note: Include jaas.jar in your classpath if you are using JDK version less than JDK1.4)

   BASE_CLASS_DIR: Set this variable to the directory where all the Sample compiled classes are located.

   JAR_DIR: Set this variable to the directory where the JAR files of the Sample compiled classes will be created.
2. In the *base-directory*`\samples\authentication\spi\providers` directory, run the `make` command.

Deploying the Login Module Sample Program

To Deploy the Login Module

1. Copy `LoginModuleSample.jar` from JAR_DIR to *AccessManager-base*`\web-src\services\WEB-INF\lib`

2. In the Web Container from which this sample has to run, update the classpath with `LoginModuleSample.jar`.

3. Update `server.xml` with the new classpath and server.xml locations:

   - Sun Java System Web Server :
     `<WS-install-dir>\https-<WS-instance-name>\config\server.xml`

   - Sun Java System Application Server:
     ```
     <AS-install-dir>\domain\<appserver domain>\<appserver_instance>
         \config\server.xml
     ```
     Example:
     `<AS-install-dir>\domain\domain1\server1\config\server.xml`

4. Copy `LoginModuleSample.xml` from *base-directory*`\samples\authentication\spi\providers` to *base-directory*`\web-src\services\config\auth\default`.

5. Restart the web container

   WebServer: `<WS-home-dir>\https-<WS-instance-name>\restart`

   Application Server: `AppServer-home-dir>\domains\`
   `<domain name>\<server_instance>\bin\restartserv`

Implementing Authentication PostProcessing SPI

The Authentication SPI includes the `AMPostAuthProcessInterface` which can be implemented for post-processing tasks. The AMPostProcessInterface Javadocs are available at:

AccessManager-base`/SUNWam/docs/com/sun/identity/authentication/spi/`
`AMPostAuthProcessInterface.html`

The SPI is configurable at the organization, service and role levels. The Authentication Service invokes the post processing SPI methods on successful or failed authentication and on logout.

About the PostProcessing SPI Sample

<PRODUCT_DIR> or *AccessManager-base* directory on different Platforms:

- **Solaris Sparc/x86:** *AccessManager-base*/SUNWam
- **Linux:** *AccessManager-base*/sun/identity

To Compile the ISAuthPostProcessSample Program on Solaris Sparc/x86 or Linux

Follow these steps given below to compile the sample found under *AccessManager-base*/`samples/authentication/spi/postprocess` .

1. Set the following environment variables.

   JAVA_HOME: Set this variable to your installation of JDK. The JDK should be version 1.3.1_06 or higher.

   CLASSPATH: Set this variable to refer to am_services.jar which can be found in the *AccessManager-base*/lib directory. (Note: Include `jaas.jar` in your classpath if you are using JDK version less than JDK1.4)

   BASE_DIR: Set this variable to the directory where the Access Manager is installed.

   BASE_CLASS_DIR: Set this variable to the directory where all the Sample compiled classes are located.

   JAR_DIR: Set this variable to the directory where the JAR files of the Sample compiled classes will be created.

   These variables will be used to run the `gmake` command. You can also set these variables in the Makefile. This Makefile is in the following directory: *AccessManager-base*/samples/authentication/spi/postprocess.

2. In the directory *AccessManager-base*/`samples/authentication/spi/postprocess,` run the `gmake` command.

To Deploy the ISAuthPostProcess Sample Program

1. Copy `ISAuthPostProcess.jar` from `JAR_DIR` to *AccessManager-base*/lib.

2. Update the Web Container configuration file `server.xml`.

   Add `ISAuthPostProcessSample.jar` to the classpath. The `server.xml` file for different web containers can be found at the following locations:

   Web Server:`<WS-home-dir>/https-<WS-instance-name>/config/`

   Application Server:`<AS-home-dir>/domain/domain1/server1/config/`

   For all other web containers consult, the manufacturer's documentation.

3. Restart the web container.

   WebServer: `<WS-home-dir>/https-<WS-instance-name>/restart`

   Application Server: `<AS-install-dir>/<domains>/<domain name>/<server instance>/bin/restartserv`
   Example: `/<AS-home-dir>/domains/domain1/server1/bin/restartserv`

   For all other web containers consult their documentation.

Configuring the Authentication Post Processing SPI

The Authentication PostProcessing Sample can be configured at the Organization,Service or Role level.

To Configure ISAuthPostProcess Sample for an Organization

1. Log in to Access Manager console as amAdmin. Use the following URL: `http://<host>.<domain>:<port>/<Console-Deploy-URI>`

2. Click Identity Management, and select your organization.

3. From the View menu, click Services.

4. In the navigation frame, under Authentication, click Core.

5. Add the following to the Authentication PostProcessing Class attribute:

   `com.iplanet.am.samples.authentication.spi.postprocess`

6. Add the following to the `Authentication PostProcessing Class` attribute:

   `ISAuthPostProcessSample`

7. Click Save.

8. Log out.

9. Go to the following URL
   `http://<host>.<domain>:<port>/<Service-Deploy-URI>/UI/Login`

If you choose to use an organization other than the default, be sure to specify that in the URL using the 'org' parameter.

The postprocessing SPI will be executed on successful authentication, on failed authentication, and on Logout.

To Configure the ISAuthPostProcess Sample for a Service

1. Log in to Access Manager console as amAdmin. Use the following URL: `http://<host>.<domain>:<port>/<Console-Deploy-URI>`
2. Click Identity Management, and select your organization.
3. From the View menu, select Services.
4. Select Authentication Configuration
5. From the Service Instance frame, select New Instance.
6. Enter a name for the service.
7. Add the following to the Authentication PostProcessing Class attribute:
   ```
   com.iplanet.am.samples.authentication.spi.postprocess.
       ISAuthPostProcessSampl
   ```
8. Click Submit to save the changes.
9. Click Service Name and define the Authentication Configuration for the new service.
10. Log out.
11. Go to the following URL:
    ```
    http://<host>.<domain>:<port>/<Service-Deploy-URI>/UI/
        Login?service=<servicename>
    ```

    If you choose to use an organization other than the default, be sure to specify that in the URL using the `org` parameter.

The postprocessing SPI will get executed on successful authentication, failed authentication and on Logout for the service accessed.

To Configure ISAuthPostProcess Sample for a Role

1. Log in to Access Manager console as amAdmin. Use the following URL: `http://<host>.<domain>:<port>/<Console-Deploy-URI>`
2. Click the Identity Management tab, and select your organization.
3. From the View menu, select Roles to view the role properties.
4. From the View menu, select Services.

5. Click Edit to edit the authentication configuration.
6. Add the following to the Authentication post Processing Class attribute:

   ```
   com.iplanet.am.samples.authentication.spi.postprocess.
       ISAuthPostProcessSample
   ```

7. Click Submit to save the changes.
8. Log out.
9. Go to the following URL:
   `http://<host>.<domain>:<port>/<Service-Deploy-URI>/UI/Login?role=roleName`

   If you choose to use an organization other than the default, be sure to specify that in the URL using the `org` parameter. Example: `org=`*orgName*

The postprocessing SPI will be executed for the service accessed on successful authentication, on failed authentication, and on Logout.

Compiling On Windows2000

Go to the *base-directory*\samples\authentication\spi\postprocess directory and run make.

To Deploy the ISAuthPostProcessSample Program

1. Copy `ISAuthPostProcess.jar` from `JAR_DIR` to *base-directory*\lib
2. In the Web Container from which this sample has to run, update the classpath with `ISAuthPostProcess.jar`.
3. Restart Access Manager.

   base-directory\bin\amserver start

To Configure Authentication Post Processing SPI

This sample can be can be set in the Core Auth Service for Organization and Authentication Configuration Service for Role OR Service.

See the section "Configuring the Authentication Post Processing SPI" on page 148.

Generating an Authentication User ID

This file explains how to compile, deploy and configure the Authentication User ID Generation SPI Sample.

- To Compile the UserIDGeneratorSample on Solaris Sparc/x86, Linux

- To Deploy the UserIDGeneratorSample program
- Configuring the UserIDGeneratorSample Program
- Compiling the UserIDGeneratorSample Program on Windows 2000

In the following sections, the PRODUCT_DIR setting depends on which platform you're using:

Solaris Sparc/x86: `PRODUCT_DIR = <install_root>/SUNWam`

Linux: `PRODUCT_DIR = <install_root>/sun/identity`

To Compile the UserIDGeneratorSample on Solaris Sparc/x86, Linux

The sample is located in the following directory:

AccessManager-base/`samples/authentication/spi/genuid`

1. Set the following environment variables.

   These variables will be used to run the `gmake` command. You can also set these variables in the Makefile which is located in the following directory:

   AccessManager-base/`samples/authentication/spi/genuid`

   JAVA_HOME: Set this variable to your installation of JDK. The JDK should be version 1.3.1_06 or higher.

   CLASSPATH: Set this variable to refer to am_services.jar which can be found in the <PRODUCT_DIR>/lib directory. (Note: Include jaas.jar in your classpath if you are using JDK version less than JDK1.4)

   BASE_DIR: Set this variable to the directory where the Access Manager is installed.

   BASE_CLASS_DIR: Set this variable to the directory where all the Sample compiled classes are located.

   JAR_DIR: Set this variable to the directory where the JAR files of the Sample compiled classes will be created.

2. In the directory *AccessManager-base*/`samples/authentication/spi/genuid`, run the `gmake` command:

To Deploy the UserIDGeneratorSample program

1. Copy `UserIDGeneratorSample.jar` from `JAR_DIR` to *AccessManager-base*/`lib`.
2. *i*n the Web Container from which this sample has to run, update the classpath with `UserIDGeneratorSample.jar`.

- On Sun ONE Web Server, go to server instance's config directory: `<WS-home-dir>/https-<WS-instance-name>/config/`
- On Sun ONE Application Server, in the directory `<AS-home-dir>/domain/domain1/server1/config/` update `server.xml` with the new classpath.
- For all other containers, consult the documentation that came with the product.

3. Restart web container.

   ```
   <WS-home-dir>/https-<WS-instance-name>/start
   ```

   ```
   <AS-home-dir>/domains/domain1/server1/bin/start
   ```

Configuring the UserIDGeneratorSample Program

The Authentication User ID Generation Sample can be configured at the Organization level, and then used or invoked by the out-of-box Membership / Self-registration authentication module.

To Configure UserIDGeneratorSample for an Organization

1. Log in to Access Manager console as amAdmin. Use the following URL: `http://<host>.<domain>:<port>/<Console-Deploy-URI>`
2. Click the Identity Management tab, and select your organization.
3. From the View menu, select Services.
4. In the navigation frame, under Authentication, click Core.
5. Add the following to the “Pluggable User Name Generator Class" attribute:

   ```
   com.iplanet.am.samples.authentication.spi.genuid.
       UserIDGeneratorSample
   ```

6. Click Save to save the changes.
7. Log out.

To Access an Authentication Module for an Organization

This module is the one which invokes the `UserIDGenerator` SPI implementation class. By default, only the Membership / Self-registration authentication module calls this SPI implementation.

1. Make sure that you have registered and enabled the Membership authentication module, and that you have created a template for the organization.

2. Enter the following URL:

   ```
   http://<host>.<domain>:<port>/<Service-Deploy-URI>/UI/
       Login?module=Membership
   ```

   If you choose to use an organization other than the default, be sure to specify that in the URL using the `org` parameter. Example: `org=`*orgName*

3. Click New User.

   You should be able to register any existing username or user ID.

The `UserIDGeneratorSample` will be executed. You will be presented with the generated User IDs choice menu to choose any one username or user ID.

Compiling the UserIDGeneratorSample Program on Windows 2000

In the `<install-root>\samples\authentication\spi\genuid` directory, run the `make` command.

To deploy the UserIDGeneratorSample Program

1. Copy `UserIDGeneratorSample.jar` from `JAR_DIR` to *<install-root> \ lib*

2. In the Web Container from which this sample has to run, update the classpath with `UserIDGeneratorSample.jar`.

3. Restart Access Manager.

   ```
   <install-root>\bin\amserver start
   ```

To Configure the UserIDGeneratorSample Program

Configuring the program on Windows 2000 is similar to configuring the program on Solaris. See "Configuring the Authentication Post Processing SPI" on page 148.

Implementing A Pure JAAS Module

A sample program demonstrates how to write pure a JAAS module to replay callbacks by authenticating using Access Manager Auth Remote API. It will authenticate a user by replaying the callbacks required by Access Manager the Authentication Module. You can modify this program to use other existing or customized Access Manager Authentication modules. This sample module can be plugged in into any standard JAAS framework using the JAAS API.

NOTE For detailed information on JAAS, see the Sun Developer Documentation at the following URL: `http://java.sun.com/products/jaas/`. For detailed information on how to write a JAAS module, see the *JAAS LoginModule Developer's Guide* at the following URL:
`http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASLMDevGuide.html`

Conventions Used in the Samples

Table 6-5 Default directories for Solaris Sparc/x86

Variable	Description	Location
Config_directory	Directory that contains configuration files	/CONFIG_DIR = /etc/opt/SUNWam/config
Product_Directory	Directory where Access Manager is installed.	PRODUCT_DIR = <install_root>/SUNWam

Table 6-6 Default directories for Linux

Variable	Description	Location
Config_Directory	Directory that contains configuration files	CONFIG_DIR = /etc/opt/sun/identity/config
Product_Directory	Directory where Access Manager is installed.	PRODUCT_DIR = <install_root>/sun/identity

Table 6-7 Default directories for Windows 2000

Variable	Description	Location
Config_Directory	Directory that contains configuration files	CONFIG_DIR = <install_root>\lib
Product_Directory	Directory where Access Manager is installed.	

To Run the Sample on Solaris Sparc x86 or Linux:

1. In the Makefile, set the following variables:

   BASE: Enter the path to the directory where Access manager is installed.

   JAVA_HOME: Enter the path to the directory where Java compiler is installed

   CONFIG: Enter the entry specified in the login configuration file. This entry will be used to do the user authentication

2. Copy `AMConfig.properties` from Access Manager server installation machine location `<CONFIG_DIR>` to the client machine where the sample will be run.

3. On the client machine, be sure the following are in your classpath:

   - `am_services.jar`
   - `jaas.jar`
   - `jss3.jar`
   - `AMConfig.properties`

   Include `jaas.jar` in your classpath if you are using a JDK version less than JDK1.4

4. A sample configuration file `purejaassample.config` is provided for testing this sample. It contains only one entry named Sample. Sample is the name to be entered for `CONFIG` in the Makefile.:

```
Sample {
 PureJAASSampleLoginModule required ORG_NAME="dc=iplanet,dc=com"
INDEX_NAME="LDAP" debug=true;
 };
```

The entry specifies that the LoginModule to be used to do the user authentication is the `PureJAASSampleLoginModule` and that this `SampleLoginModule` must succeed in order for authentication to be considered successful. It passes options with `ORG_NAME` as the organization name and `INDEX_NAME` as the Access Manager authentication module to which this sample must authenticate.

If you must use a different login configuration, modify the Makefile. For example, change the following:

`-Djava.security.auth.login.config=purejaassample.config`

to this:

`-Djava.security.auth.login.config=<yourj_jaas_config_file.config>`

5. To compile, run the `gmake` command.
6. To run the sample program run the `gmake run` command.

To Enable SSL

1. In the sample client program, add this JVM property:

   `-D "java.protocol.handler.pkgs=com.iplanet.services.comm"`

2. In the `AMConfig.properties` file, edit the following properties:

   com.iplanet.am.admin.cli.certdb.dir: `<PRODUCT_DIR>/servers/alias`

   com.iplanet.am.admin.cli.certdb.prefix: `https-machine1.com-machine1-`

   com.iplanet.am.server.protocol: `https`

   com.iplanet.am.server.port: Enter the appropriate port on the server machine where `machine1` is the host name of the server

To Run the Sample on Windows 2000

1. In `make.bat`, set the following properties:

   BASE: Enter the path to the directory where Access manager is installed

   JAVA_HOME: Enter the path to the directory where the Java compiler is installed.

   CONFIG: Enter the entry which will be used for user authentication. This entry is specified in the login configuration file.

2. Copy `AMConfig.properties` from Access Manager server installation machine location `<CONFIG_DIR>` to the client machine where this sample will be run.
3. On the client machine, make sure the following are in your classpath:

   ❍ `am_services.jar`

   ❍ `jaas.jar`

   ❍ `jss3.jar`

   ❍ `AMConfig.properties`

   Include `jaas.jar` in your classpath if you are using JDK version less than JDK1.4.

4. A sample configuration file `purejaassample.config` is provided for testing this sample. It contains only one entry named.Sample. Sample is the name to be entered for `CONFIG` in the Makefile.

```
Sample {
 PureJAASSampleLoginModule required ORG_NAME="dc=iplanet,dc=com"
INDEX_NAME="LDAP" debug=true;
 };
```

   The entry specifies that the LoginModule to be used to do the user authentication is the `PureJAASSampleLoginModule.` SampleLoginModule is must succeed in order for authentication to be considered successful. It passes options with `ORG_NAME` as the organization name and `INDEX_NAME` as the Access Manager authentication module to which this sample has to authenticate.

   If you must use a different login configuration, modify the Makefile. For example, change the following:

   `-Djava.security.auth.login.config=purejaassample.config`

   to this:

   `-Djava.security.auth.login.config=<yourj_jaas_config_file.config>`

5. To compile, run the `make` command.

6. To run the sample program, run the `make run` command.

To Enable SSL

1. In the sample client program, add this JVM property:

   `-D "java.protocol.handler.pkgs=com.iplanet.services.comm"`

2. Edit the following properties in the `AMConfig.properties` file:

   com.iplanet.am.admin.cli.certdb.dir:
   `<ISinstall-dir>\SUN\IdentityServer6\Servers\alias`

   com.iplanet.am.admin.cli.certdb.prefix:
   `https-machine1.red.iplanet.com-machine1-`

   com.iplanet.am.server.protocol: `https`

   com.iplanet.am.server.port: Enter the appropriate port on the server machine where `machine1` is the host name of the server

For the detailed information, see the Javadoc for Remote Client APIs. By default, Access Manager Javadoc is installed in the following directory:

AccessManager-base/`SUNWam/docs`

For the detailed information on how to plug the Login Module into the standard JAAS Context, see the *JAAS Reference Guide* at `http://java.sun.com/j2se/1.5.0/docs/guide/security/jaas/JAASRefGuide.html`.

Chapter 7

Identity Management

The Identity Management module of Sun Java™ System Access Manager 6 2005Q1 contains an XML template file and application programming interfaces (APIs) that provide the functionality to, among other operations, create, delete and manage identity entries in the Sun Java System Directory Server used for data storage. This chapter offers information on these identity-related features. It contains the following sections:

- "Overview" on page 159
- "Identity-related Objects" on page 160
- "Object Templates And ums.xml" on page 164
- "amEntrySpecific.xml" on page 168
- "Identity Management SDK" on page 169
- "Identity Management Samples" on page 183

Overview

The Identity Management module allows for the management of Identity-related Objects using the Access Manager console or command line tools. These objects, that are created and managed via Access Manager, are actually stored as LDAP entries in Directory Server. To bridge the gap between the two products, Access Manager provides interfaces that are used to create and delete identity-related objects as well as get, add, modify, or remove their attributes.

Access Manager Console

All aspects of the Access Manager console are covered in Chapter 3, "The Access Manager Console," of this manual and the *Sun Java System Access Manager Administration Guide*.

ums.xml

This file defines a set of *templates* that contain the configuration information needed to set up each identity-related object created with Access Manager as an LDAP entry in the Directory Server data store. More information on `ums.xml` can be found in "Object Templates And ums.xml" on page 164.

Identity Management Software Development Kit (SDK)

The SDK is used to integrate the management functions of Access Manager into external applications or services. More information on the SDK can be found in "Identity Management SDK" on page 169.

Identity-related Objects

Access Manager defines and manages the following identity-related objects:

- Organizations
- Containers
    - Organizational Units (referred to as *containers* in the console)
    - People Containers
    - Group Containers
- Users
- Groups
    - Static Groups
    - Assignable Groups (Dynamic)

    - Filtered Groups
- Roles
    - Static Roles
    - Filtered Roles

These identity-related objects are not LDAP objects as defined in the Directory Server schema. These objects are configured using an Access Manager schema, managed using the Access Manager application and only stored in Directory Server. In other words, an identity-related object in Access Manager does not necessarily correspond to its LDAP counterpart in Directory Server. But, because they are stored in Directory Server, these Access Manager objects must be mapped to the existing Directory Server schema. Thus, Access Manager object entries are appended with *marker object classes*.

Marker Object Classes

An identity-related object stored in Directory Server is identified as such through the use of special marker object classes appended to its LDAP entry. These object classes are defined in the Access Manager schema, `ds_remote_schema.ldif`, located in *IdentityServer_base*`/SUNWam/ldif`. When a marker object class is added to a Directory Server entry, Access Manager is able to access and manage that entry using its console or command line tools. For example, an enterprise's existing directory schema may use *organizational unit* as its root rather than the default *organization*; by adding the Access Manager *organization* marker object class, `sunManagedOrganization`, to the LDAP entries of the organizational unit, Access Manager can manage it as the organization's root. It is through the use of marker object classes that Access Manager can manage most existing directory structures. The marker object classes are:

- `sunManagedOrganization`
- `iplanet-am-managed-org-unit`
- `iplanet-am-managed-people-container`
- `iplanet-am-managed-group-container`
- `iplanet-am-managed-person`
- `iplanet-am-managed-static-group`
- `iplanet-am-managed-group`
- `iplanet-am-managed-assignable-group`

- `iplanet-am-managed-filtered-group`
- `iplanet-am-managed-role`
- `iplanet-am-managed-filtered-role`

For information on how to configure an existing directory tree within Access Manager, see the *Sun Java System Access Manager Migration Guide*.

Identity-related Objects As LDAP Entries

Following is a discussion of the Access Manager objects and how they map to LDAP entries in Directory Server.

Organizations

Represented by the marker object class `sunManagedOrganization`, organization is the root entry of an Access Manager tree. It generally maps to an LDAP `organization` or `organizationalUnit` object class.

Containers

Functionally, there are three types of containers in Access Manager.

Organizational Units

Represented by the marker object class `iplanet-am-managed-org-unit`, an organizational unit is referred to as a *container* in the Access Manager console. It generally maps to the LDAP `organizationalUnit` object class and can contain sub-organizations, other containers, roles, groups, and users.

People Containers

Represented by the marker object class `iplanet-am-managed-people-container`, a people container is an organizational unit which is a parent for user entries. It generally maps to the LDAP `organizationalUnit` object class and can contain sub-people containers and users.

Group Containers

Represented by the marker object class `iplanet-am-managed-group-container`, a group container is an organizational unit which is a parent for any number of group entries. It generally maps to the LDAP `organizationalUnit` object class and can only contain groups and other group containers.

Users

Represented by the markerobject class `iplanet-am-managed-person`, a user is the representation of a person. It maps to an LDAP `inetOrgPerson`. It is a leaf node that may not contain other entries.

Groups

Functionally, there are three types of groups in Access Manager. Assignable Groups (Dynamic) (by default) and Static Groups are configured using the Membership By Subscription option in the console. Filtered Groups are configured by choosing the Membership By Filter option in the console.

Assignable Groups (Dynamic)

Represented by the marker object class `iplanet-am-managed-assignable-group`, an assignable group is one in which an administrator wants to explicitly add the user to a group. For example, Larry wants to give Ramona permission to look at his employees' telephone numbers so he adds her to the ReadPhoneNumbers group. In Directory Server, member entries contain the `memberof` LDAP attribute (`inetAdmin` object class) and the group membership is dynamically established.

NOTE Assignable groups are referred to as Dynamic when seen in the console as, technically, they are created with an LDAP filter albeit a static one.

Static Groups

Represented by the marker object class `iplanet-am-managed-static-group`, a static group is one in which members are added by appending the `groupOfUniqueNames` object class to the LDAP group entry itself. It can contain users, filtered groups or other static sub-groups. This type of group can be enabled using the Administration Service in the console. By default, it is disabled and all groups created are of the type "Assignable Groups (Dynamic)."

Filtered Groups

Represented by the marker object class `iplanet-am-managed-filtered-group`, a filtered group is created through the use of an LDAP filter. All user entries are funneled through the filter and dynamically assigned to the group. The filter would look for a specified attribute in an entry and return those entries that contain the attribute as a member of the group.

Roles

Functionally, there are two types of roles in Access Manager. Roles can only be created in organizations, suborganizations and generic containers; they can not be configured in people containers.

Static Roles

Represented by the marker object class `iplanet-am-managed-role`, a static role is a role entry in which the members are added by appending the `groupOfUniqueNames` object class to the role entry itself. It can contain users.

Filtered Roles

Represented by the marker object class `iplanet-am-managed-filtered-role`, a filtered role is created through the use of an LDAP filter. All user entries are funneled through the filter and dynamically assigned to the role. The filter would look for a specified attribute in an entry and return those entries that contain the attribute as a member of the role.

Object Templates And ums.xml

The `ums.xml` provides a set of parameters, or *templates*, that contain the LDAP configuration information for all Identity-related Objects managed using Access Manager. The templates are used to create LDAP entries for the identity-related objects so they can be stored in Directory Server. In addition, the templates are used for the dynamic generation of roles and the construction of object searches. The file can be found in the *IdentityServer_base*/`SUNWam/config/ums` directory; it is based on the `sms.dtd` which is defined in Chapter 8, "Service Management," of this manual.

NOTE These templates can be modified by administrators to alter the behavior of the Java interfaces. But, if `ums.xml` is modified and reloaded, there will be inconsistencies between the entries created prior to the modifications and the newer ones.

Structure Of ums.xml

The `ums.xml` defines three types of templates: Structure, Creation and Search. Structure templates define the Directory Server information tree attributes for the object. Creation templates define an LDAP template for the object being created. Search templates define guidelines for performing searches using LDAP.

Structure Templates

Structure templates define the form an Access Manager object will take within the Directory Server information tree. In other words, these templates define the child nodes (roles, groups, containers) that are created IN ADDITION to the creation of the object itself. There are six attributes that need to be defined for each object’s structure.

- `class`—This attribute represents the name of the Java class that will implement the object. This attribute is fixed and should never be modified.
- `name`—This attribute defines the Relative Distinguished Name (RDN) for the object. RDN is "ou=People" or "cn=ContainerDefaultTemplateRole". For the core structure templates such as Organization or OrganizationaUnit, the value defined at run time (when you create Org's or containers from console or CLI). Thats why you don't see the RDN value for the core ones. Where as for others such as PeopleContainer & DefaultOrgRole, you see the RDN's. You can specify the RDN values for the PeopleContainers, Groups that can be created. A note of caution that the naming attribute specified in the RDN, for example ou from ou=Groups should match the naming attribute defined in the Group Creation template. For example, an organization has `o=org` as its naming attribute while a people container uses `ou=People`.
- `childNode`—This attribute specifies the child nodes (roles, groups, containers) that will be created in tandem with the object. The value is the name of the structure template for the respective object.
- `template`—This attribute specifies the name of the Creation template used to create this object.
- `filter`—This attribute is not currently used.
- `priority`—For internal use only, the value of this attribute should always remain 0.

Creation Templates

Every identity object that Access Manager creates has a corresponding creation template which defines the LDAP schema for the object. It specifies which object classes and attributes are mandatory or optional and which default values, if any, should be set. This conforms to the actual LDAP entry in the Directory Server. There are six attributes that need to be defined for each object’s template.

- `name`—This attribute defines the type of object that the template will create. It is also the name of the template itself. This attribute should not be modified.

- `javaclass`—This attribute defines the name of the Java class used to instantiate the object. This attribute should not be modified.
- `required`—This attribute defines the required LDAP object classes and attributes for the object.
- `optional`—This attribute defines the optional LDAP object classes and attributes for the object.
- `validated`—This attribute is reserved for future use.
- `namingattribute`—This attribute specifies the LDAP attribute used to name the object. For instance, the Basic User creation template has as its `namingattribute` the value of the LDAP attribute, `uid`.

Search Templates

Search templates are used to define how searches for identity-related objects are performed in Directory Server. This template defines a default search filter and the attributes returned in the search. For example, a search filter is constructed which defines and specifies which attributes and values are to be retrieved from the Directory Server.

- `name`—This attribute defines the name of the search template.
- `searchfilter`—This attribute defines the value the search will look for.
- `attrs`—This attribute specifies the LDAP attributes that need to be returned.

NOTE For a listing of interfaces applicable to each identity-related objects, see "amEntrySpecific.xml" on page 168.

Modifying ums.xml

Any LDAP attributes or object classes not already present in the Directory Server LDAP schema must be added to the `ums.xml` file in order for them to be recognized by the Access Manager. In most cases, the attributes that service developers might want to add may already exist in the `inetorgperson` and the `inetuser` object classes. If, for example, a custom mail service is being added with, specifically, an `employeeNumber` attribute, the `ums.xml` file does not need to be modified because this attribute already exists in the `inetorgperson` object class. Generally, the `ums.xml` file does not need to be modified. Some circumstances where this file would need to be modified are:

- if Access Manager is being installed against a legacy DIT.

- if new object classes are being added to users or organizations.
- if service developers want to change the default organizations or roles.
- if service developers need to change an entry's naming attribute.

Additional information on when and how to modify the `ums.xml` file is covered in the section on installing against a legacy DIT in the *Sun Java System Access Manager Migration Guide*.

CAUTION It is recommended that the `ums.xml` configuration file be backed up before any modifications are made.

Adding Custom Object Classes

If a service developer wants to add new or customized object classes to the Directory Server for Access Manager's use, they would need to modify the templates in the `ums.xml` file. The DAI Service would then need to be deleted from Directory Server and the modified ums.xml reloaded using the `amadmin` command line tool.

Once `ums.xml` has been modified, the new object classes and attributes must be defined in an XML service file which would then be imported into Access Manager using the procedures described in Chapter 8, "Service Management," of this manual. This configures Access Manager to manage the new object classes from the console.

NOTE `umsExisting.xml` contains objectclasses and user object class tags which will be replaced after installation and is used when installing Access Manager with an existing directory server information tree.

DAI Service

When Access Manager is installed, the `ums.xml` file is stored in Directory Server as the Directory Access Instructions (DAI) service. The DAI service is only available for modification through the Directory Server; it is not available through the Access Manager console or command line interface. The Access Manager SDK gets the configuration information from this directory tree node, when needed, to create an identity-related object or perform a search. Any attribute specified in the `ums.xml` can be set for a created object. If `ums.xml` is modified, the DAI Service would need to be deleted from Directory Server and reloaded using the `amadmin` command line tool. To delete the DAI Service from Directory Server, delete the DAI branch (`ou=DAI,ou=services,`*root-suffix*) or use the `amadmin` command line tool with the -*r* option. To reload `ums.xml`, use `amadmin` and the -*s* option. (The administrator user and password options will also be used for both.) For more detailed information on the command line tools, see the *Sun Java System Access Manager Administration Guide*.

NOTE When using the `amadmin` command line tool to reload `ums.xml`, the full DN of the `amadmin` user must be used as a parameter. If not, the LDAP Authentication Service will not be able to find the administrator in its search for the user DN. For example, instead of using `amadmin -u amadmin -w 11111111` -s *ums.xml file path,* the input command would be:

```
amadmin -u
"uid=amadmin,ou=people,dc=example_org,dc=com" -w
11111111 -s ums.xml file path
```

amEntrySpecific.xml

The purpose of the `amEntrySpecific.xml` service file is to define attributes from an existing directory to display on the Access Manager console's functional pages for all Identity-related Objects. These functional pages are as follows:

- Create—The Create page is displayed when the administrator clicks `New`.
- Properties—The Properties Page is displayed when the Properties icon (an arrow in a box) next to an object is clicked.
- Search—The Search link is in the top left frame of the Access Manager console.

Each object can have its own schema definition in the `amEntrySpecific.xml` file which is based on the `sms.dtd` as described in Chapter 8, "Service Management," of this manual.

NOTE Dynamic attributes are not supported in `amEntrySpecific.xml`.

If a service developer wants to customize the console's functional pages with attributes that are not default to the Access Manager tree, they would modify the `amEntrySpecific.xml` file. For example, to display an attribute on the group page, the new attribute needs to be added to the `amEntrySpecific.xml` file. Any object with customized attributes in the Directory Server would need to have those attributes reflected in the `amEntrySpecific.xml` file also. (Most often, a service developer would only be customizing the organization pages.) Code Example 7-1 is the organization attribute subschema that defines the display of an organization's Organization Status and its choice values. Note that based on the information in"any Attribute" on page 208, this Organization Status attribute will be displayed on the Search page and is not an attribute requiring a value for creation.

Code Example 7-1 Organization Subschema of `amEntrySpecific.xml`

```
...
<SubSchema name="Organization">
                    <AttributeSchema name="inetdomainstatus"
                         type="single_choice"
                         syntax="string"
                         any="optional|filter"
                         i18nKey="o2">
                         <ChoiceValues>
                             <ChoiceValue>Active</ChoiceValue>
                             <ChoiceValue>Inactive</ChoiceValue>
                         </ChoiceValues>
                    </AttributeSchema>
                </SubSchema>
...
```

If the `type` attribute is not specified in `amEntrySpecific.xml`, the defaults will be used. A default setting means that only the name of the entry will display on the object function pages in the Access Manager console.

All the attributes listed in the schema definitions in the `amEntrySpecific.xml` file are displayed when the abstract type object pages are displayed. If the attribute is not listed in a schema definition in the `amEntrySpecific.xml` file, the Access Manager console will not display the attribute.

NOTE The User service is not configured in the `amEntrySpecific.xml` file but in its own `amUser.xml` file.

Identity Management SDK

The Access Manager SDK contains an API for identity management. These interfaces can be used by developers to integrate management functions into external applications or services that will be managed by Access Manager. The API functions to create or delete identity-related objects as well as get, modify, add or delete the object's attributes. The `com.iplanet.am.sdk` package contains all the interfaces and classes necessary to perform these operations in Directory Server.

Interfaces

Below are brief explanations of the Identity Management API.

NOTE All operations performed using the API open and close LDAP connections via a connection pool. The connection pool size can be set in the serverconfig.xml file. For more information, see Appendix B, "serverconfig.xml File," in this manual.

AMAssignableDynamicGroup

The `AMAssignableDynamicGroup` interface provides the methods used to manage "Assignable Groups (Dynamic)." This class extends the base `AMGroup` interface. Associated with this object are the following `ums.xml` templates that define its behavior at runtime. The creation template used is the *BasicAssignableDynamicGroup*; and the search template used is the *BasicAssignableDynamicGroupSearch*. It does not have a pre-defined structural template.

AMCallback

`AMCallBack` is a plug-in class that needs to be extended by external applications in order to do special pre / post-processing for the creation, deletion and modification operations for users, organizations, roles and groups.

AMConstants

`AMConstants` is the base interface for all identity-related objects. It is used to define constants for use with the SDK (constants associated with searches, etc.).

AMDynamicGroup

The `AMDynamicGroup` interface provides the methods used to manage dynamic groups. This class extends the base `AMGroup` interface. Associated with this object are the following `ums.xml` templates that define its behavior at runtime. The creation template used is named *BasicDynamicGroup*; and the search template used is named as *BasicDynamicGroupSearch*. It does not have a pre-defined structural template.

AMEventListener

The `AMEventListener` interface that can be used to monitor and react to events. This *listener* can be called when an identity-related object is removed, renamed or modified. It must be implemented using the following procedure:

1. Implement the `AMEventListener` interface.
2. Get an instance of the object to which `AMEventListener` will listen.

   For example, get an `AMUser` object and add the listener: `AMUser.addEventListener()`.
3. When an event changes something in this object, the listener will be called.

CAUTION Access Manager does not currently support attaching an event listener to template creation code.

AMFilteredRole

The `AMFilteredRole` interface provides the methods used to manage "Filtered Roles." Associated with this object are the following `ums.xml` templates that define its behavior at runtime. The creation template used is *BasicFilteredRole*; and the search template used is *BasicFilteredRoleSearch*. It does not have a pre-defined structural template.

AMGroup

The `AMGroup` interface provides the methods used to manage groups. This is the basic class for all derived groups, such as static groups, dynamic groups and assignable dynamic groups. No default templates are defined for this class.

AMGroupContainer

The `AMGroupContainer` interface provides the methods used to manage "Group Containers." Associated with this object are the following `ums.xml` templates that define its behavior at runtime. The structural template used by this class is *GroupContainer*; the creation template used is *BasicGroupContainer*, and the search template is *BasicGroupContainerSearch*.

AMObject

`AMObject` provides basic methods to manage identity-related objects. Since this is a generic class, it does not have any templates (as defined in "Object Templates And ums.xml" on page 164) associated with it.

AMOrganization

The `AMOrganization` interface provides the methods used to manage "Organizations." Associated with this interface are the following `ums.xml` templates that define its behavior at runtime. The structural template used by this class is *Organization*; the creation template used is *BasicOrganization*, and the search template is *BasicOrganizationSearch*.

NOTE The `AMOrganization` interface contains methods that can be used to search through identity-related objects in Directory Server. More information can be found in "Search Methods In The SDK" on page 175.

AMOrganizationalUnit

The `AMOrganizationalUnit` interface provides the methods used to manage "Organizational Units." Associated with this object are the following `ums.xml` templates that define its behavior at runtime. The structural template used by this class is *OrganizationalUnit*; the creation template used is *BasicOrganizationalUnit*, and the search template is *BasicOrganizationalUnitSearch*.

AMPeopleContainer

The `AMPeopleContainer` interface provides the methods used to manage "People Containers." Associated with this object are the following `ums.xml` templates that define its behavior at runtime. The structural template used by this class is *PeopleContainer*; the creation template used is *BasicPeopleContainer*, and the search template is *BasicPeopleContainerSearch*.

AMRole

The `AMRole` interface provides the methods used to manage "Roles." Associated with this object are the following `ums.xml` templates that define its behavior at runtime. The creation template used is *BasicManagedRole*; and the search template used is *BasicManagedRoleSearch*. It does not have a pre-defined structural template.

AMSearchControl

The `AMSearchControl` class provides a way to customize search behavior. Common behaviors are time limit, result limit and virtual list view.

Code Example 7-2 Sample Code Using AMSearchControl

```
SSOTokenManager manager = SSOTokenManager.getInstance();
SSOToken token = manager.createSSOToken(new
AuthPrincipal("uid=amadmin,ou=People,dc=example,dc=com"), "11111111");
suo = getSampleUserOperations(token);
amsc = new AMStoreConnection(token);
//System.out.println(suo.createUser(amsc));
AMSearchControl amc = new AMSearchControl();
amc.setTimeOut (2000);
amc.setSearchScope (AMConstants.SCOPE_ONE);
AMPeopleContainer amp =
amsc.getPeopleContainer("ou=people,dc=example,dc=com");
Set userset = (amp.searchUsers(amc, "(uid=u*)")).getSearchResults();
Object users[] = userset.toArray();
System.out.println((String)users[0]);
System.exit(0);
```

AMStaticGroup

The `AMStaticGroup` interface provides the methods used to manage "Static Groups." This class extends the base `AMGroup` interface. The name of the creation template used with this class is *BasicGroup*; and the search template used is *BasicGroupSearch*. It does not have a pre-defined structural template.

AMStoreConnection

The `AMStoreConnection` class provides the means to establish a connection to the data store Directory Server and provides methods to create, remove and get different types of identity-related objects. A `SSOToken` is required in order to instantiate a `AMStoreConnection` object.

AMTemplate

The `AMTemplate` interface represents a service template associated with `AMObject`. Access Manager distinguishes between virtual and entry attributes. As defined for Sun Java System Directory Server, a *virtual attribute* is an attribute not physically stored in an LDAP entry but still returned with it as a result of a LDAP search. Virtual attributes are analogous to *inherited* attributes. An *entry attribute* is a non-inherited attributes.

NOTE More information on virtual attributes can be found in "Virtual Attribute" on page 373 of Appendix E, "Directory Server Concepts," in this manual.

For AMOrganization, AMOrganizationalUnit and AMRole, virtual attributes can be grouped in a template on a per-service basis; there may be one service template for each service for any given AMObject. Such templates determine the service attributes inherited by the users within the scope of this object. The templates are: DYNAMIC_TEMPLATE and ORGANIZATION_TEMPLATE. DYNAMIC_TEMPLATE are implemented using CoS; ORGANIZATION_TEMPLATE does not have virtual attributes or LDAP attributes.

Template Priority

When an object inherits more than one template for the same service (by virtue of being in the scope of two or more objects with service templates), the conflict is resolved through template priorities. (This conflict will only occur with services that contain "Dynamic Attributes.") The priority is defined by the value of the "cosQualifier Attribute" as discussed in Chapter 8, "Service Management," of this manual. (The comparison values are default, override, and merge-schemes.) The priority level for a service template is set when then template is created using the Access Manager console. The levels are Highest, Higher, High, Medium, Low, Lower, and Lowest. Templates with higher priorities will be favored over templates with lower priorities when default is the value of cosQualifier. In the case where two or more templates are being considered for inheritance of an attribute value, and they have the same (or no) priority, the result is merged. If the value is override, the priority level of the template takes precedence over any priority specified in the user profile. Merge-schemes signifies that the priority values will not be used, but a merged list of attribute values from all templates will be assigned. Templates which do not have an explicitly assigned priority are considered to have the lowest priority possible, or no priority.

AMUser

The AMUser interface provides the methods used to manage "Users." Associated with this object are the following ums.xml templates that define its behavior at runtime. The creation template used is *BasicUser*; and the search template used is *BasicUserSearch*. It does not have a pre-defined structural template.

Default Implementation Of AMUser

There is a default implementation of AMUser. Assuming an SSOToken and a user DN, the code to find the user status is illustrated in Code Example 7-3.

Code Example 7-3 Sample Code To Find User Status

```
AMStoreConnection conn = new AMStoreConnection (ssoToken) ;
AMUser user = conn.getUser (userDN) ;
if (user.isActivated()) {
....
```

Code Example 7-3 Sample Code To Find User Status *(Continued)*

```
} else {
....
}
```

AMUserPasswordValidation

`AMUserPasswordValidation` is an interface to plugin external modules to validate user names and passwords. The methods of this class must be overridden by the implementation plugin modules. The modules will be invoked whenever a userID or password value is being added or modified using Access Manager console, the `amadmin` CLI or the SDK directly.

Search Methods In The SDK

The SDK provides a variety of methods to conduct searches throughout the organizational tree. They are provided within the AMOrganization interface. Criteria is needed by the API to perform a search. Typically, the criteria is a LDAP search filter string, the scope of the search (one level or sub-tree), and where the search will begin (the base DN). The SDK provides the APIs to conduct searches and obtain results for all identity objects.

NOTE The SDK always includes the objectclass used to search so it is not required to explicitly include the filter. For example if searching for users, the SDK will include the default user search filter provided in the BasicUserSearch search template in the ums.xml.

This section specifically discusses one of the search methods: `searchUsers`. (For information on all of the search methods, refer to the Access Manager Javadocs.) Code Example 7-4 is the set of different search methods available for `searchUsers`.

Code Example 7-4 Available Search Methods For searchUsers

```
    public Set searchUsers(String wildcard, int level)
        throws AMException, SSOException;

    public Set searchUsers(String wildcard, Map avPairs, int level)
        throws AMException, SSOException;
```

Code Example 7-4 Available Search Methods For searchUsers

```
     public AMSearchResults searchUsers(String wildcard, Map avPairs,
AMSearchControl searchControl)
         throws AMException, SSOException;

     public AMSearchResults searchUsers(String wildcard,
 AMSearchControl searchControl)
         throws AMException, SSOException

     public AMSearchResults searchUsers(String wildcard,
 AMSearchControl searchControl, String avfilter)
         throws AMException, SSOException;
```

Search Method Parameters

Here are brief descriptions of some of the search method parameters.

AMSearchControl

This class provides a way to specify detailed search criteria such as the scope of the search, the maximum results, time out value, etc. It must be implemented for all searches to set these criteria.

wildCard

This parameter can be used to specify the wild card used for naming attributes. For example, if searching for all users whose naming attributes (uid or cn) start with "Ma", then the wild card could be Ma*.

avPair

This parameter is a map of attribute/value pairs that need to be added to a search filter. The key of the map is the attribute name and the value is a set of values. The SDK will construct a filter from this `avPair` map. Each of the pairs in the map will be OR ("|") and not AND (&) to construct the filter.

avFilter

In most cases it will be sufficient to OR the attributes, but this parameter provides flexibility for applications to pass their own search filter to meet search criteria. Such filters could be a complex LDAP search filter as in the following example:

```
(&(objectclass=iplanet-am-managed-person)((customEmployeeNumer=12*)
&(customDepartment=3459932)))
```

This example illustrates when two conditions (the employee number and department number) need to be met. For this purpose, AND (&) is used.

NOTE The methods that return a `java.util.Set` will throw an exception if the search fails as a result of exceeding the search limit or the time limit. In such cases, even partial results of the failed search will not be returned. To obtain the partial results in such cases, the methods that return an `AMSearchResults` object must be used. The error code can be verified by using the class methods to check if the search was successful.

searchUsers Sample Code

Code Example 7-5 demonstrates how to search for all users in an organization (DN: `dc=example,dc=com`) who belong to department *3459932* and whose user names end with *smith*.

Code Example 7-5 Sample Code For Search Methods

```
// Note obtain a valid token of a principal who has privileges to
// perform this operation.
SSOToken token = getSSOToken();

// Create an AMStoreConnection and obtain an AMOrganization
// instance for dc=example, dc=com
AMStoreConnection amc = new AMStoreConnection(token);
AMOrganization amOrg = amc.getOrganization("dc=example,dc=com");

// Construct the search filter
// Need to retrieve all usernames ending with smith
String wildCard = "*smith"
Map avPair = new HashMap();
Set departmentValue = new HashSet();
departmentValue.add("3459932");
avPair.put("customDepartment", departmentValue);

// Set the search control
AMSearchControl = new AMSearchControl();
// Sub tree search
searchControl.setSearchScope(AMConstants.SCOPE_SUB);
// Time out 3000 milliseconds.
searchControl.setTimeOut(3000);
// Would like to get only first 100 results
searchControl.setMaxResults(100);

// Perform the search
AMSearchResults results = amOrg.searchUsers(wildcard, avPair,
                                             searchControl);
// Check if any time out or size limit errors occured.
if (results.getErrorCode == AMSearchResults.SUCCESS) {
     // Process the results
} else {
     // Verify the error condition and take appropriate action
}
```

Here the filter to conduct the search will look like:

`(&(uid=*smith)(objectclass=inetorgperson)((customerDepartment="3459932")))`

To add an additional department, one more value can be added to the search as in:

`(&(uid=*smith)(objectclass=inetorgperson)((customerDepartment="3459932")|(customerDepartment="3459933")))`

Search Groups Sample Code

Code Example 7-6 uses interfaces from the `com.iplanet.am.sdk` package to search groups.

Code Example 7-6 Search Groups Code Sample

```
try {
    Set orgSet1 = new HashSet();
    Set orgSet2 = new HashSet();
    Set orgSet3 = new HashSet();
    Set orgSet4 = new HashSet();
    AMSearchResults results = null;
    AMSearchControl ctl = new AMSearchControl(); //use default values
    String DN = "ou=Groups,dc=idp1,dc=com";
    AMOrganizationalUnit org = conn.getOrganizationalUnit(DN);
    if (org.isExists()) {
    //get all groups in this OU:
        orgSet1 = org.getAssignableDynamicGroups(AMConstants.SCOPE_SUB);
//get Assignable Dynamic Groups
       orgSet2 = org.getDynamicGroups(AMConstants.SCOPE_SUB); //get Dynamic
Groups
        orgSet3 = org.getStaticGroups(AMConstants.SCOPE_SUB); //get Static
Groups

    //set up the avPairs for the search on attribute within group
        Map avPairs = new HashMap();
        Set set = new HashSet(1);
        set.add("true");
        avPairs.put("iplanet-am-group-subscribable", set);
        results = org.searchAssignableDynamicGroups( "*", avPairs, ctl );
//returns all subscribable groups
        orgSet4 = results.getSearchResults();
    }
    //Print the results
    return "Assignable Dynamic Groups: " + orgSet1.toString() +
            "Dynamic Groups: " + orgSet2.toString() +
            "Static Groups: " + orgSet3.toString() +
            "Group with subscribable=true:" + orgSet4.toString();
    } catch (Exception ex) {
        ex.printStackTrace();
```

Code Example 7-6 Search Groups Code Sample

```
    return "got errors";
    }
```

Email Notification And The SDK

`amProfile.properties` is the localization file for the SDK. All strings that may be visible via an error message or a feature are stored in this file as `key=value` pairs. The file itself is located in *IdentityServer_base*/`SUNWam/locale`. Although all of the properties are not discussed in this section, there are some worth noting that pertain to email notification. The Administration Service has a number of notification attributes: User Creation, User Deletion and User Modification notification lists. When a user profile is created, deleted or modified, a notification email will be sent to the addresses listed as values of these attributes. To modify the message that is sent, the following `key=value` pairs in `amProfile.properties` need to be modified.

- 490=The user creation email subject can be defined with this key. The default is `WARNING: user creation notice`.
- 491=The user deletion email subject can be defined with this key. The default is `WARNING: user deletion notice`.
- 492=The user modification email subject can be defined with this key. The default is `WARNING: user modification notice`.
- 493=The user creation email body text can be defined with this key. The default is `user is created:` followed by the DN of the user.
- 494=The user deletion email body text can be defined with this key. The default is `user is deleted:` followed by the DN of the user.
- 495=The user modification email body text can be defined with this key. The default is `user is modified:` *user_DN*. attribute is changed: attribute old_value: *original_value* new value: *modified_value*
- 497=The entity from which the email comes is defined with this key. The default is Identity-Server.

More information on the Administration Service and the notification attributes themselves can be found in the *Sun Java System Access Manager Administration Guide*.

Caching And The SDK

Caching in the Identity Management SDK is used for storing all `AMObject` attributes (For example, attributes of identity-related objects) that are retrieved from Directory Server. The cache does not hold `AMObject` directly, only its attributes. All attributes retrieved from Directory Server using the methods `AMObject.getAttributes()`, `AMObject.getAttribute(String name)` or `AMObject.getAttributes(setAttributeNames)` will be cached. Table 7-1 contains a listing of the recorded cache properties.

Table 7-1 Recorded Cache Properties

Information Name	What is recorded
Number of requests during this interval	Number of get requests during the specified interval
Number of cache hits during this interval	Number of hits during the specified interval
Hit ratio for this interval	Hit ratio for the specified interval
Total number of requests since server start	Overall number of get requests since a server re-start
Total number of cache hits since server start	Overall number of hits since a server re-start
Overall Hit ratio	Overall hit ratio since a server re-start
Total Cache Size	The total size of the cached information

Cache properties can be configured by modifying attributes in the `AMConfig.properties` file. For more information see "SDK Caching" on page 338 of Appendix A, "AMConfig.properties File," in this manual.

Installing The SDK Remotely

It is possible for an external application to perform management functions on the Directory Server without installing the full Access Manager application at the external location. By installing the `SUNWamsdk` package using the `pkgadd` utility (or the installer), the Identity Management SDK can be installed on a non-Access Manager machine. For more details on the Identity Management SDK only installation option, refer to the *Java Enterprise System Installation Guide*.

NOTE If the `SUNWamsdk` package is installed remotely and Access Manager is running in SSL mode, a certificate database needs to be created. Create the database using the Sun Java System Web Server command line tool `certutil` or the Web Server console and then copy the database to the remote machine. For more information, see the Sun Java System Web Server documentation set.

Management Function Samples

Following are several samples that illustrate identity management functions using the Identity Management SDK.

NOTE Access Manager can authenticate and authorize against directories other than Sun Java System Directory Server (for example, Microsoft™ Active Directory), but Access Manager can not perform management functions against these directories such as creating users or deleting organizations.

Creating Objects

Typically, three steps are involved in creating an object with the SDK. The following three steps are specific to creating users but can be modified for any object.

To Create A User

1. Get `AMStoreConnection` object to connect to the data store.
2. From the `AMStoreConnection`, get `AMPeopleContainer` object where the users will be created.
3. In `AMPeopleContainer` object, create users.

Code Example 7-7 Sample Code To Create A User

```
/**
   * This method will describe the SDK usage for creating a user.
   * It uses AMStoreConnection to get the organization object
   * It also uses the Set Parameters to store the different
   * attributes of the user. It throws
   * an AMException if it's unble to create it and we throw
   * message "unable to create" to the GUI by catching the same
   */

    public String createUser(HttpServletRequest req, Set parameters,
AMStoreConnection conn) {
        try {
```

Code Example 7-7 Sample Code To Create A User *(Continued)*

```
            Map userAttributeMap = new HashMap();
            if (parameters.contains("uid")) {
                uid = req.getParameter("uid");
                storeUserAttributes("uid", uid, userAttributeMap);
            }
            if(parameters.contains("firstname")) {
                firstName = req.getParameter("firstname");
                storeUserAttributes("givenname", firstName,
userAttributeMap);
            }
            if(parameters.contains("lastname")) {
                lastName = req.getParameter("lastname");
                storeUserAttributes("sn", lastName, userAttributeMap);
            }
            if(parameters.contains("password")) {
                passWord = req.getParameter("userPassword");
                storeUserAttributes("userPassword", passWord,
userAttributeMap);
            }

            Map userMap1 = new HashMap();
            userMap1.put(uid, userAttributeMap);
            String orgDN = req.getParameter("orgName");
            String dn = "ou=People" + "," + orgDN;
            AMPeopleContainer ampc = conn.getPeopleContainer(dn);
            ampc.createUsers(userMap1);
            userDN = "uid=" + uid + "," + dn;
            /*
             * This is to keep the context of the user
             */
            contextUser = conn.getUser(userDN);
            return showCreateUserSuccess();
        } catch (Exception ex) {
            ex.printStackTrace();
          return "Unable to create";
    }
```

To Create An Organization

1. Get `AMStoreConnection` object to connect to the data store.
2. From the `AMStoreConnection`, get `AMOrganization` object for the top level organization.
3. In `AMOrganization` object, create sub-organization.

NOTE `org.createUsers` creates users directly under the organization. In order to create users in a people container, use the `AMPeopleContainer` object.

Retrieve Templates

Code Example 7-8 retrieves a service's dynamic templates by opening a connection to Directory Server with `AMStoreConnection`. It retrieves a service's dynamic template by defining the DN of the top organization (`toporg.com`) as well as the string attribute of the specific service to be retrieved.

Code Example 7-8 Retrieve Service's Dynamic Template

```
...
     // instantiate a store connector from SSO Token
     AMStoreConnection  amsc = new AMStoreConnection(ssoToken);
     // retrieve top level organization by DN
     AMOrganization org = amsc.getOrganization("dc=toporg,dc=com");
     // retrieve Dynamic  type AMTemplate for iPlanetAMSessionService
     AMTemplate template = org.getTemplate("iPlanetAMSessionService",
AMTemplate.DYNAMIC_TEMPLATE);
     // retrieve attributes
     String maxSessionTime =
template.getStringAttribute("iplanet-am-session-max-session-time");
     ...
```

TIP As an alternative to creating a new XML service file, `amUser.xml` can be modified. In this case, unregister the old `amUser` service file, modify it and re-register the modified file. Attribute/value pairs need to be integrated into the `amUser.properties` file for newly-defined internationalization keys. `ums.xml` does not need to be modified for this option.

Identity Management Samples

Access Manager contains samples that illustrate user management functions. These include a sample to add an attribute to the user profile and one to illustrate how to create organizations, users, roles, and services using the SDK. They can be found in *IdentityServer_base*`/SUNWam/samples/um`.

Adding User Attributes

This sample explains how to add new attributes to the User profile so that those new attributes can be managed via the user page in the Access Manager console. There are 2 ways this can be achieved: modify the existing `amUser.xml`, or create a new XML service file and import it into Access Manager.

Creating Objects With The SDK

This sample contains sample Java code that can be generated and run to create some identity-related objects including an organization, roles and users. The defined `SampleOrgOperations.java` creates an organization, gets the registered services, and adds them. `SampleUserOperations.java` and `SampleRoleOperations.java` can also be used for their respective purposes.

Chapter 8

Service Management

Sun Java™ System Access Manager 6 2005Q1 provides a mechanism for the definition and management of services and their configuration data. Both eXtensible Markup Language (XML) files and Java™ interfaces are used for this purpose. This chapter provides information on how to define a service, the structure of the XML files and the service management application programming interfaces (API). It contains the following sections:

- "Overview" on page 185
- "Defining A Custom Service" on page 187
- "DTD Files" on page 198
- "XML Service Files" on page 231
- "Service Management SDK" on page 239

Overview

A *service* is a group of attributes that are managed together by the Access Manager console. The attributes can be the *configuration parameters* of a software module or they might just be related information with no connection to a software application. As an example of the first scenario, after creating a payroll module, a developer can create an XML service file that might include attributes to define an employee name, an hourly pay rate and an income tax rate. This XML file is then integrated into the Access Manager deployment so that these three attributes and their corresponding values can be stored in, and managed from, the Sun Java System Directory Server data store and Access Manager console, respectively.

Access Manager provides the necessary tools for administrators to define, integrate and manage groups of attributes as a service. Creating a service for management using the Access Manager console involves preparing an XML service file, configuring an LDAP Data Interchange Format (LDIF) file with any new object classes and importing both, the XML service file and the new LDIF schema, into the Directory Server. Administrators can then register, customize and manage the service using the Access Manager console. More specific information on this process can be found in "Defining A Custom Service" on page 187.

NOTE Throughout this chapter, the term *attribute* is used to illustrate two concepts. An Access Manager or service attribute refers to the configuration parameters of a defined service. An XML attribute refers to the parameters that qualify an XML element in an XML service file.

XML Service Files

XML service files enable Access Manager to manage attributes that are stored in Directory Server. It is important to remember that Access Manager does not implement any behavior or dynamically generate any code to interpret the attributes; it can only set or get the attribute values. Out-of-the-box though, Access Manager loads a number of services it uses to manage the attributes of its own features; it manages and uses these values. For example, the Logging attributes are displayed and managed in the Access Manager console, while code implementations within the Access Manager use these configured attributes to record the operations of the application. All XML service files are located in `/etc/opt/SUNWam/config/xml`. For more specific information on the XML files used in service management, see "XML Service Files" on page 231.

NOTE Any application with LDAP attributes can have its data managed using the Access Manager console by configuring a custom XML service file and loading it into the Directory Server. For more information, see "Defining A Custom Service" on page 187.

Document Type Definition Structure Files

The format of an XML file is based on a structure defined in a DTD file. In general, a DTD file defines the elements and qualifying attributes needed to write a well-formed and valid XML document. Access Manager exposes the DTD files that are used to define the structure for the different types of XML files it uses. The

DTDs are located in *IdentityServer_base*/`SUNWam/dtd`. This chapter primarily concerns itself with `sms.dtd`, the file that defines the structure for all XML service files. Additional information on Access Manager DTDs can be found in "DTD Files" on page 198.

NOTE Knowledge of XML is necessary to understand DTD elements and how they are integrated into Access Manager. When creating an XML file, it might be helpful to print out the relevant DTD and a corresponding sample XML file.

Service Management SDK

Access Manager also provides a service management SDK that gives application developers the interfaces necessary to register and un-register services as well as manage schema and configuration information. These interfaces are bundled in a package called `com.sun.identity.sm`. More information on the SDK can be found in "Service Management SDK" on page 239.

Defining A Custom Service

To define a service for management using Access Manager, the developer must create an XML service file as well as configure an LDIF file for any object classes not already defined in Directory Server. Both, the XML service file and the new LDIF schema, must then be imported into Directory Server. Once imported, the service can be registered to an organization using Access Manager and its attributes managed and customized by the Access Manager administrator. The following steps detail the procedure used to define a service. The sections following the procedure explain each step in more detail.

1. Create an XML service file containing a group of attributes.

   This XML service file must conform to the `sms.dtd`. A simple way to create a new XML service file would be to copy and modify an existing one. More information on creating an XML service file can be found in "Creating A Service File" on page 189. An explanation of the DTD syntax can be found in "The sms.dtd Structure" on page 199.

2. Extend the LDAP schema in Directory Server using `ldapmodify`, if necessary.

   Loading an LDIF file into Directory Server will add any newly defined or modified LDAP object classes and attributes to the directory tree. This step is only necessary when defining dynamic, policy and user attributes. (Using Access Manager-specific object classes and attributes do not require that changes be made to the LDAP schema.) Instructions on extending the LDAP schema can be found in "Extending The Directory Server Schema" on page 193. Additional information on identity-related objects and the Access Manager schema can be found in Chapter 7, "Identity Management," of this manual and the *Sun Java System Access Manager Deployment Planning Guide*, respectively. The Sun Java System Directory Server documentation contains information on the LDAP schema.

3. Import the XML service file into Directory Server using `amadmin`.

   Information on importing an XML service file and the `amadmin` command line utility can be found in "Importing The XML Service File" on page 195 and the *Sun Java System Access Manager Administration Guide*, respectively.

4. Configure a localization properties file and copy it into the *IdentityServer_base*/`SUNWam/locale` directory.

   The localization properties file must be created with accurate `i18nKey` fields. These console names map to fields defined in the XML service file. If no localization properties file exists, Access Manager will display the actual attribute names. More information on the localization properties file can be found in "Configuring Console Localization Properties" on page 196 and "Localization Properties Files" on page 90 of Chapter 5, "Authentication Service," in this manual.

5. Update the `amEntrySpecific.xml` or `amUser.xml` files, if necessary.

   The `amEntrySpecific.xml` file defines the attributes that will display on the Create, Properties and Search pages specific to each of the Access Manager *abstract objects*. The `amUser.xml` file can be modified to add User attributes to the User Service. (Alternately, User attributes can be defined in the actual XML service file in which case, `amUser.xml` would not need to be modified.) Information on abstract objects and updating `amEntrySpecific.xml` can be found in Chapter 7, "Identity Management," of this manual. Information on modifying `amUser.xml` can be found in "Modifying A Default XML Service File" on page 233.

6. Register the service using Access Manager console.

After importing the service into Directory Server, it can be registered to an organization and the attributes managed through the Access Manager console. Information on how this can be done is in the Service Configuration chapter in the *Sun Java System Access Manager Administration Guide*. Information on how to register the service using the command line can be found in "Registering The Service" on page 197.

Creating A Service File

The information in this section corresponds to Step 1 on page 187, creating an XML service file. The XML service file defines the attributes of an Access Manager service. It must follow the structure defined in the `sms.dtd` which enforces the service developer to combine attributes into one of five groups, allowing the developer to differentiate between those attributes applicable to, for example, a service instance or a user. The DTD syntax can be found in "The sms.dtd Structure" on page 199.

Service File Naming Conventions

When creating a new XML service file, there are some naming conventions that must be followed.

- The name of a service (other than an authentication module service) as defined in the XML service file can be any string as long as it is unique.
- The name of an authentication module service as defined in the XML service file must be in the form `iPlanetAMAuth`*module_name*Service.)
- Any defined authentication level attribute must be configured as `iplanet-am-auth-`*module_name*`-auth-level`.

Service Attributes

The `sms.dtd` requires the service developer to define attributes into one of five groups. These groups differentiate between those attributes applicable to, for example, the Access Manager deployment as a whole, a specific service or a single user.

Global Attributes

Global attributes are defined for the entire Access Manager installation and are common to all data trees, service instances and integrated applications within the configuration. Global attributes can not be applied to users, roles or organizations as their purpose is to configure Access Manager itself. Server names, port numbers, service plug-ins, cache size, and maximum number of threads are examples of global attributes that are configured with one value. For example, when Access Manager performs logging functions, the log files are written into a directory. The location of this directory is defined as a global attribute in the Logging Service and all Access Manager logs, independent of their purpose, are written to it. Access Manager administrators can modify these default values using the console. Global attributes are stored in Directory Server using specially-defined LDAP attributes so the LDAP schema does not need to be extended to add a new global attribute.

NOTE If a service has only global attributes, it can not be registered to an organization nor can a service template be created. An example of this would be the Platform Service.

Organization Attributes

Organization attributes are defined and assigned at the organization level. Attributes for an Authentication Service are a good example. When the Authentication Service is registered, attributes are configured depending on the organization to which it is registered. The `LDAP Server` and the `DN To Start User Search` would be defined at the organization level as this information is dependent on the address of an organization's LDAP server and the structure of their directory tree, respectively. Organization attributes are stored in Directory Server using specially-defined LDAP attributes so the LDAP schema does not need to be extended to add a new organization attribute.

NOTE Organization attributes are not inherited by sub-organizations. Only dynamic attributes can be inherited. For additional information, see "Attribute Inheritance" on page 192.

Dynamic Attributes

Dynamic attributes are *inheritable* attributes that work at the role and organization levels as well as the sub-organization and organizational unit levels. Services are assigned to organizations and roles which, in general, have access to any service assigned to its parent organization. Dynamic attributes are inherited by users that possess a role or belong to the organization. Because dynamic attributes are assigned to roles or organizations instead of set in a user entry, they are *virtual* attributes inherited by users using the concept of *Class of Service* (CoS). When these attributes change, the administrator only has to change them once, in the role or organization, instead of a multitude of times in each user entry.

NOTE Dynamic attributes are modeled using *class of service* (CoS) and *roles*. For information on these features, see Appendix E, "Directory Server Concepts," in this manual or refer to the Sun Java System Directory Server documentation.

An example of a dynamic attribute might be the address of a common mail server. Typically, an entire building might have one mail server so each user would have a mail server attribute in their entry. If the mail server changed, every mail server attribute would have to be updated. If the attribute was in a role that each user in the building possessed, only the attribute in the role would need to be updated. Another example might be the organization's address. Dynamic attributes are stored within the Directory Server as LDAP objects, making it feasible to use traditional LDAP tools to manage them. A Directory Server LDAP schema needs to be defined for these attributes.

Policy Attributes

Policy attributes specify the access control actions (or *privileges*) associated with a service. They become a part of the rules when rules are added to a policy. Examples include `canForwardEmailAddress` and `canChangeSalaryInformation`. The actions specified by these attributes can be associated with a resource if the `IsResourceNameAllowed` element is specified in the attribute definition. For example, in the web agent XML service file, `amWebAgent.xml`, `GET` and `POST` are defined as policy attributes with an associated URL resource as `IsResourceNameAllowed` is specified.

NOTE Out of the box, only the Policy Configuration Service uses policy attributes although they can be defined for any number of services.

User Attributes

User attributes are defined for a single user. User attributes are not inherited from the role, organization, or sub-organization levels. They are typically different for each user, and any changes to them would affect only the particular user. User attributes could be an office telephone number, a password or an employee ID. The values of these attributes would be set in the user entry and not in a role or organization. For example, if 70 attributes are user-defined and an organization has two million users, each attribute is stored two million times. This, of course, only occurs if the service is assigned to the user and a value is set for them. User attributes can be a part of any service but, for convenience, Access Manager has grouped a number of the most widely-used attributes into a service defined by the `amUser.xml` service file. User attributes are stored within the Directory Server as LDAP objects, making it feasible to use traditional LDAP tools to manage them. A Directory Server LDAP schema needs to be defined for these attributes.

NOTE When defining user attributes in an XML service file (other than `amUser.xml`), the service must be assigned to the user for the user attributes to be displayed on their User Profile page. In addition, the User Profile Display Option in the Administration Service must be set to `Combined`. For more information, see the *Sun Java System Access Manager Administration Guide.*

Attribute Inheritance

After creating and loading an XML service file, an administrator can assign the service's attributes by registering it and creating a service template. Then, when a user possesses a role or belongs to an organization to which the service is registered, they inherit the dynamic attributes of the role or the service, respectively. Inheritance only occurs, though, when the service possessed is explicitly assigned to the user. A user can inherit attributes from multiple roles or parent organizations.

TIP Service templates created for a parent organization contain attributes that *trickle down* to sub-organizations. Therefore it is not necessary to create templates for sub-organizations unless the attribute values are being customized. Creating a large number of service templates will have a performance impact.

ContainerDefaultTemplateRole Attribute

Dynamic attributes are used in an XML service file if an administrator wants to define a particular attribute as one which is inherited by all identity objects to which the service is registered. After uploading the XML service file and registering the service to an organization or role, all users in the sub-trees of the organization or role will inherit the dynamic attributes. To accomplish this, Access Manager uses *classic CoS* and role templates (as described in Appendix E, "Directory Server Concepts"). `ContainerDefaultTemplateRole` is a default *filtered* role configured for each organization in which the LDAP object class `iplanet-am-managed-person` is the default filter. Every user in Access Manager is a member of `iplanet-am-managed-person` so every user in the organization possesses `ContainerDefaultTemplateRole`. Access Manager creates a separate

CoS template for each registered service which points to the service's dynamic attributes. Because of this, any user who has `ContainerDefaultTemplateRole` (all of them, by default) will inherit the dynamic attributes of the service. The LDIF entry for `ContainerDefaultTemplateRole` is illustrated in Code Example 8-1.

Code Example 8-1 `ContainerDefaultTemplateRole` LDIF Entry

```
dn: cn=ContainerDefaultTemplateRole,o=example
objectClass: top
objectClass: nscomplexroledefinition
objectClass: nsfilteredroledefinition
objectClass: nsroledefinition
objectClass: ldapsubentry
nsRoleFilter: (objectclass=iplanet-am-managed-person)
```

Modifying Inheritance

The `nsRoleFilter` attribute (as displayed in Code Example 8-1 may be modified to allow objects other than users to inherit from `ContainerDefaultTemplateRole`. Formatting its value as, for example, `(|(objectclass=iplanet-am-managed-person)(objectclass=organization))` allows users and organizations to inherit the dynamic attributes. Any valid filter syntax can be used although typically it would be limited to attributes or objectclasses in the user entries. In addition, the relevant objectclass from the LDAP attributes must also be added to the entry.

Extending The Directory Server Schema

The information in this section corresponds to Step 2 on page 188, extending the LDAP schema in Directory Server. When configuring an XML service file for Access Manager, it might also be necessary to modify the Directory Server schema. First, any customized dynamic, policy or user attributes defined in an Access Manager service that are not already defined in the Directory Server schema need to be associated with an LDAP object class. Then the attribute(s) and object class(es) need to be added to the LDAP schema using the `ldapmodify` command line tool and an LDIF file as input.

NOTE The order in which the LDAP schema is extended or the XML service file is loaded into Directory Server is not important.

To Extend The Directory Server LDAP Schema

1. Create an LDIF file to define any new or modified LDAP object classes and attributes.

2. Change to the Access Manager `bin` directory.

   `cd` *IdentityServer_base*`/SUNWam/bin`

3. Run `ldapmodify` using the LDIF file as input.

   The syntax is `ldapmodify -D` *userid_of_DSmanager* `-w` *password* `-f` *path_to_LDIF_file*. By default, *userid_of_DSmanager* is `cn=Directory Manager`. If the LDIF was created correctly, the result of this command would be to modify the entry `cn=schema`.

NOTE After extending the schema, it is not necessary to restart the Directory Server but, as `ldapmodify` is server-specific, the schema needs to be extended on all configured servers. Information on how this is done can be found in the Sun Java System Directory Server documentation.

4. Run `ldapsearch` to ensure that the schema has been created.

   The syntax is `ldapsearch -b cn=schema -s base -D` *userid_of_DSmanager* `-w` *password* `(objectclass=*) | grep -i servicename`. If the LDIF was created correctly, the result of this command would be a listing of the object classes as illustrated in Code Example 8-2.

Code Example 8-2 Sample LDIF Listing For Mail Service

```
objectClasses: ( 1.2.NEW
 NAME 'am-sample-mail-service'
 DESC 'SampleMail Service' SUP top AUXILIARY
 MAY ( am-sample-mail-service-status $
 am-sample-mail-root-folder $
 am-sample-mail-sentmessages-folder $
 am-sample-mail-indent-prefix $
 am-sample-mail-initial-headers $
 am-sample-mail-inactivity-interval $
 am-sample-mail-auto-load $
 am-sample-mail-headers-perpage $
 am-sample-mail-quota $
 am-sample-mail-max-attach-len $
 am-sample-mail-can-save-address-book-on-server )
 X-ORIGIN ’user defined’ )
attributeTypes: ( 11.24.1.996.1
 NAME ’am-sample-mail-service-status’
 DESC ’SampleMailService Attribute’
```

Code Example 8-2 Sample LDIF Listing For Mail Service

```
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
X-ORIGIN 'user defined' )
```

Adding Access Manager Object Classes To Existing Users

If a new service is created and the service's users already exist, the service's object classes need to be added to the user's LDAP entries. To do this, Access Manager provides migration scripts for performing batch updates to already-existing user entries. No LDIF file need be created. These scripts and the procedures are described in the *Sun Java System Access Manager Migration Guide*. Alternatively, registered services can be added to each user by selecting the service on their Properties page although, for an organization with many users, this would be time-consuming.

CAUTION It is not recommended to use `ldapmodify` to extend the schema.

Importing The XML Service File

The information in this section corresponds to Step 3 on page 188, importing an XML service file into Access Manager. This step is important as it serves to populate Directory Server and Access Manager with the newly defined service attributes.

1. Change to the Access Manager install directory:

   `cd` *IdentityServer_base*`/SUNWam/bin`

2. Run following command line application: `./amadmin --runasdn` *DN_of_directory_server_administrator* `--password` *password_directory_server_administrator* `--verbose --schema` *xml_service_file_path*.

   More information on the `amadmin` command line tool can be found in the *Sun Java System Access Manager Administration Guide*

NOTE If changing an existing service, the original XML service file must be deleted before importing the newly modified XML service file. Information on this function can be found in the Sun Java System Directory Server documentation.

Configuring Console Localization Properties

The information in this section corresponds to Step 4 on page 188, configuring a localization properties file. A localization properties file specifies the locale-specific screen text that an administrator or user will see when directed to a service's attribute configuration page.

NOTE For certain services, this file also localizes error messages, Java exceptions and email notification specifics. This section though concerns itself only with service-related values. Additional information can be found in "Localization Properties Files" on page 90 of Chapter 5, "Authentication Service," in this manual.

The localization properties files are located in the *IdentityServer_base*/`SUNWam/locale` directory. They are generally named using the format `am`*service_name*`.properties`. Code Example 8-3 is the localization properties file for the Client Detection service named `amClientDetection.properties`.

Code Example 8-3 amClientDetection.Properties File

```
...
# attr descriptions msgs
#
iplanet-am-client-detection-service-description=Client Detection
a100=Client Types
a101=Default Client Type
a102=Client Detection Class
a103=Client Detection Enabled
a100.link=Edit
unknown_key=requested key is not available in the property
null_key=null key passed to getProperty
null_clientType=client type is null
unknown_clientType=requested clientType doesn't exist
update_error=notification received between setproperty and store. Need to do
setproperty again.
```

The localization properties files consist of a series of key=value pairs. The value of each pair will be displayed on the service's Properties page in the Access Manager console. The keys (`a1`, `a2`, etc.) map to the `i18nKey` fields defined for each attribute in a service in the XML service file. The keys also determine the order in which the fields are displayed on screen as the keys are displayed in the order of their ASCII characters (a1 is followed by a10, followed by a2, followed by b1). For example, if an attribute needs to be displayed at the top of the service attribute page, the alphanumeric key should have a value of `a1`. The second attribute could then have a value of either `a10`, `a2` or `b1`, and so forth.

TIP If a localization properties file is modified, Access Manager needs to be restarted to see the changes. If importing a new localization properties file, Access Manager does not need to be restarted.

Localizing With Two Languages

When one instance of Access Manager is localized with two languages, the localization properties files still go into the same directory. Each file name would be appended with a suffix to match the locale. For example, if French localization packages are added, the file name would be `am`*service_name*`_fr.properties`. If Spanish localization packages are added, that properties file name would be `am`*service_name*`_es.properties`.

NOTE Information on downloading and installing localized versions of Access Manager can be found at `http://wwws.sun.com/software/download/inter_ecom.html`.

Updating Files For Abstract Objects

For information corresponding to Step 5 on page 188, updating the `amEntrySpecific.xml`, see Chapter 7, "Identity Management," of this manual. For information corresponding to Step 5, updating the `amUser.xml`, see "XML Service Files" on page 231.

Registering The Service

The information in this section corresponds to Step 6 on page 189, registering a new service to an identity object. The preferred way to register a service is to use the Access Manager console. Information on how this is done can be found in the *Sun Java System Access Manager Administration Guide*. An alternate process to register a service is to use the `amAdmin.dtd`, batch processing templates and the command line. Information can be found in "The amAdmin.dtd Structure" on page 209 and "Batch Processing With XML Templates" on page 234.

NOTE To register a service, ensure that Access Manager is properly binding to the Directory Server.

DTD Files

Access Manager contains numerous DTD files to define the structures for the XML files used in Access Manager. The DTDs are located in *IdentityServer_base*`/SUNWam/dtd` and include:

- `Auth_Module_Properties.dtd`—defines the structure for XML files used by each authentication module to specify the properties for the Authentication Service interface. Information on this document can be found in "Authentication Programming Interfaces" on page 155 in Chapter 5, "Authentication Service," of this manual.
- `amAdmin.dtd`—which defines the structure for XML files used to perform batch LDAP operations on the directory tree using the command line tool `amAdmin`. Information on this document can be found in "The amAdmin.dtd Structure" on page 209.
- `amWebAgent.dtd`—defines the structure for XML files used to handle requests from, and send responses to, web agents. This file is deprecated and remains for purposes of backward compatibility.
- `policy.dtd`—defines the structure for XML files used to store policies in Directory Server. Information on this document can be found in the Access Manager *Administration Guide*.
- `remote-auth.dtd`—defines the structure for XML files used by the Authentication Service's remote Authentication API. Information on this document can be found in "The remote-auth.dtd Structure" on page 137 of Chapter 5, "Authentication Service," of this manual.
- `server-config.dtd`—defines the structure for `serverconfig.xml` which details ID, host and port information for all server and user types. Information on this document can be found in Appendix B, "serverconfig.xml File," in this manual.
- `sms.dtd`—which defines the structure for XML service files. Information on this document can be found in "The sms.dtd Structure" on page 199.

- web-app_2_2.dtd—defines the structure for XML files used by the Access Manager deployment container to deploy J2EE applications. The corresponding XML file is called a *deployment descriptor* which specifies container options and describes specific configuration requirements to be resolved by the deployer.

CAUTION None of the DTD files should be modified. The APIs and their internal parsing functions are based on the installed definitions. Any alterations to the DTD files will hinder the operation of Access Manager.

The sms.dtd Structure

The sms.dtd defines the data structure for all XML service files. It is located in the *IdentityServer_base*/SUNWam/dtd directory. The sms.dtd enforces the developer to define each service attribute as one of five types which are then stored and managed differently. For instance, some of the attributes are applicable to an entire Access Manager installation (such as a port number or server name), while others are applicable only to individual users (such as a password). The attribute types are Global, Organization, Dynamic, Policy, and User. More information on these types can be found in "Service Attributes" on page 189.

An explanation of the main elements defined by the sms.dtd follows. Each element includes a number of XML attributes which are also explained. Explanations of the remaining elements can be found in the sms.dtd file itself. Access Manager currently supports only about some of the elements contained in sms.dtd; this section discusses only those elements.

NOTE Customized attribute names in XML service files should be written in lower case as Access Manager converts all attribute names to lower case when reading from the Directory Server.

ServicesConfiguration Element

ServicesConfiguration is the root element of the XML service file. It allows for the definition of multiple services per one XML file. Its immediate sub-element is the Service Element. Code Example 8-4 on page 200 illustrates the *ServicesConfiguration* element as defined in the amClientDetection.xml service file located in /etc/opt/SUNWam/config/xml.

Code Example 8-4 ServicesConfiguration and Service Element

```
...
<ServicesConfiguration>
<Service name="iPlanetAMClientDetection" version="1.0">
<Schema...>
...
```

Service Element

The *Service* element defines the schema for one given service. A number of different services can be defined in one XML file using this element, although this is not recommended. Currently, Access Manager supports the following sub-elements: Schema Element (which defines the service's attributes as either *Global*, *Organization*, *Dynamic*, *User* or *Policy*) and *Configuration*. The required XML attributes for the *Service* element are the name of the service, such as *iPlanetAMLogging*, and the version number of the XML service file itself. Code Example 8-4 on page 200 also illustrates the *Service* element, its attributes and the opening Schema tag.

Schema Element

The *Schema* element is the parent of the family of elements that define the service's attributes and their default values. The sub-elements can be the Global Element, Organization Element, Dynamic Element, User Element or Policy Element. The required XML attributes of the *Schema* element include the serviceHierarchy Attribute, the i18nFileName Attribute, the i18nKey Attribute, and the propertiesViewBeanURL Attribute.

serviceHierarchy Attribute

When a new service is configured, its name will be dynamically displayed in the Navigation frame of the console based on the value of this attribute. The value is a "/" separated string. Each "/" portion of the string represents a level in the hierarchy. Code Example 8-5 on page 201 illustrates the `serviceHierarchy` attribute as defined in `amClientDetection.xml`. `iPlanetAMClientDetection` is the name of the service. The name used for display in the console, though, is defined by the `i18nKey` (or i18nKey Attribute), and retrieved from the service's localization file defined by the i18nFileName Attribute. In this example, the value of `iplanet-am-client-detection-service-description` will be found in `amClientDetection.properties` and its value displayed. The service name will be displayed below the Access Manager Configuration header in the left frame of the Service Configuration module. To prevent a service from displaying in the console, either remove the `serviceHierarchy` attribute or set it's value to `""`, as in `serviceHierarchy=""`.

NOTE `DSAMEConfig` as used in Code Example 8-5 and all XML service files refers to the Access Manager Configuration header. The use of DSAME is a holdover from the previous name of Access Manager. This is defined in the `amAdminModuleMsgs.properties` file located in *IdentityServer_base*/`SUNWam/locale`.

Code Example 8-5 `i18nFileName`, `i18nKey` and `serviceHierarchy` Attributes

```
...
<Schema
 serviceHierarchy="/DSAMEConfig/iPlanetAMClientDetection"
           i18nFileName="amClientDetection"
 i18nKey="iplanet-am-client-detection-service-description">
...
```

i18nFileName Attribute

The `i18nFileName` attribute refers to the localization properties files. It takes a value equal to the name of the localization properties file for the defined service (minus the `.properties` file extension). For example, Code Example 8-5 defines the name of the properties file as `amClientDetection`.

i18nKey Attribute

The value of the `%i18nIndex` attribute maps to the final, localized name of the service to be displayed in the Access Manager console as it is defined in the localization properties file.

NOTE The `%i18nIndex` attribute is defined as an *entity* at the top of the `sms.dtd`. In the configured XML service files, `%i18nIndex` is replaced by `i18nKey` and its corresponding value.

For example, Code Example 8-5 refers to the value of the iplanet-am-client-detection-service-description attribute as defined in `amClientDetection.properties`. This value is the name of the service as it will be displayed in the Access Manager console; in this case, **Client Detection** is the name defined in `amClientDetection.properties`. (Remember, the value of the defined attribute might not be in English.) More information on the localization properties file can be found in Chapter 5, "Authentication Service," of this manual.

NOTE If the `i18nKey` value is blank (`i18nKey=""`), the Access Manager console will not display the attribute.

propertiesViewBeanURL Attribute

The default display for a service is a simple table showing the attribute name and its value. The `propertiesViewBeanURL` attribute provides the URL to the Java bean used by the console to generate this display. It is possible to override the default display by creating a new class and defining the URL to this class as a value of this attribute. If no value is specified, the display is created by the console.

Service Attribute Elements

The next five elements are sub-elements of the "Schema Element" on page 200; they are the declarations of the service's Access Manager attributes. When defining a service, each attribute must be defined as either a Global Element, an Organization Element, a Dynamic Element, a User Element, or a Policy Element. Any configuration of these elements (all of them or none of them) can be used depending on the service. Each attribute defined within these elements is itself defined by an AttributeSchema Element.

Global Element

The Global element defines Access Manager attributes that are modifiable on a platform-wide basis and applicable to all instances of the service in which they are defined. They can define information such as port number, cache size, or number of threads, but Global elements also define a service's LDAP object classes. For additional information, see "Global Attributes" on page 190.

serviceObjectClasses Attribute. The `serviceObjectClasses` attribute is a global attribute defined in an XML service file that contains either dynamic or user elements (attributes). The value of this attribute is an object class set in the LDAP entries (stored in Directory Server) for users whom are registered to the service. It allows any user with this object class to be dynamically assigned the service's dynamic or user attributes, if any exist.

CAUTION If the `serviceObjectClasses` attribute is not specified and the service has defined dynamic or user attributes, an object class violation is called when an administrator tries to create a user under that organization, and assign this service.

Multiple values can be defined for the `serviceObjectClasses` attribute. For example, if a service is created with two attributes each from three other services, the `serviceObjectClasses` attribute would need to list all three object classes as `DefaultValues`. Code Example 8-6 illustrates a `serviceObjectClasses` attribute with a defined object class from `amClientDetection.xml`.

Code Example 8-6 serviceObjectClass Defined As Global Element

```
...
<Global>
            <AttributeSchema name="serviceObjectClasses"
                type="list"
                syntax="string"
                i18nKey="">
                <DefaultValues>
 <Value>iplanet-am-client-detection-service</Value>
                </DefaultValues>
            </AttributeSchema>
 </Global>
 ...
```

Organization Element

The Organization element defines Access Manager attributes that are modifiable per organization or sub-organization. For example, a web hosting environment using Access Manager would have different configuration data defined for each organization it hosts. A service developer would define different values for each organization attribute *per* organization. These attributes are only accessible using the Access Manager SDK. For additional information, see "Organization Attributes" on page 190.

Dynamic Element

The Dynamic element defines Access Manager attributes that can be inherited by all user objects. Examples of Dynamic elements would be user-specific session attributes, a building number, or a company mailing address. Dynamic attributes use the Directory Server features, CoS and roles. For additional information, see "Dynamic Attributes" on page 190.

User Element

The User element defines Access Manager attributes that exist physically in the user entry. User attributes are not inherited by roles or organizations. Examples include password and employee identification number. They are applied to a specific user only. For additional information, see "User Attributes" on page 191.

Policy Element

The Policy element defines Access Manager attributes intended to provide actions (or *privileges*). This is the only attribute element that uses the `ActionSchema` element to define its parameters as opposed to the `AttributeSchema` element. Generally, privileges are `GET`, `POST`, and `PUT`; examples of privileges might include `canChangeSalaryInformation` and `canForwardEmailAddress`. For additional information, see "Policy Attributes" on page 191.

SubSchema Element

The `SubSchema` element can specify multiple sub-schemas of global information for different defined applications. For example, logging for a calendar application could be separated from logging for a mail service application. The required XML attributes of the *SubSchema* element include `name` which defines the name of the sub-schema, `inheritance` which defines whether this schema can be inherited by one or more nodes on the directory tree, `maintainPriority` which defines whether priority is to be honored among its peer elements, and "i18nKey Attribute" on page 201.

NOTE The `SubSchema` element is used only in the `amEntrySpecific.xml` file. It should not be used in any external XML service files.

AttributeSchema Element

The *AttributeSchema* element is a sub-element of the five schema elements discussed in "Service Attribute Elements" on page 202 as well as the SubSchema element described in "SubSchema Element" on page 204. It defines the structure for each configurable parameter (or attribute) of a service. The sub-elements that qualify the AttributeSchema can include `IsOptional?`, `IsServiceIdentifier?`, `IsResourceNameAllowed?`, `IsStatusAttribute?`, `ChoiceValues?`, `BooleanValues?`, `DefaultValues?`, or `Condition`. The XML attributes that define each portion of the attribute value are the "name Attribute", the "type Attribute", the "uitype Attribute", the "syntax Attribute", the "cosQualifier Attribute", `rangeStart`, `rangeEnd`, `minValue`, `maxValue`, `validator`, the "any Attribute", the “propertiesViewBeanURL Attribute” on page 202 and, the “i18nKey Attribute” on page 201. Code Example 8-7 on page 205 illustrates an *AttributeSchema* element taken from amUser.xml, its attributes and their corresponding values. Note that this example attribute is a Dynamic attribute.

Code Example 8-7 AttributeSchema Element With Attributes

```
...
<Dynamic>
  <AttributeSchema name="iplanet-am-user-login-status"
                type="single_choice"
                syntax="string"
                any="display"
                i18nKey="d105">
                <ChoiceValues>
 <ChoiceValue i18nKey="u200">Active</ChoiceValue>
 <ChoiceValue i18nKey="u200">Inactive</ChoiceValue>
                </ChoiceValues>
                <DefaultValues>
                    <Value>Active</Value>
                </DefaultValues>
 </AttributeSchema>
...
```

name Attribute

This required XML attribute defines the a name for the attribute. Any string format can be used but attribute names must be in lower-case. Code Example 8-7 on page 205 defines it with a value of iplanet-am-user-login-status.

type Attribute

This attribute specifies the kind of value the attribute will take. The default value for type is `list` but it can be defined as any one of the following:

- `single` specifies that the user can define one value.
- `list` specifies that the user can define a list of values.
- `single_choice` specifies that the user can choose a single value from a list of options. A default value must be defined from the list.
- `multiple_choice` specifies that the user can choose multiple values from a list of options. A default value must be defined from the list.

ChoiceValues Sub-Element. If the `type` attribute is specified as either `single_choice` or `multiple_choice`, the ChoiceValues sub-element must also be defined in the `AttributeSchema` element. Depending on the type specified, the administrator or user would choose either one or more values from the choices defined. The possible choices are defined in the ChoiceValues sub-element, ChoiceValue. Code Example 8-7 on page 205 defines the `type` as `single_choice`; the `ChoiceValues` attribute defines the list of options as `Active` and `Inactive` with the DefaultValue as `Active`.

syntax Attribute

The `syntax` attribute defines the format of the value. The default value for syntax is `string` but, it can be defined as any one of the following:

- `boolean` specifies that the value is either true or false.
- `string` specifies that the value can be any string.
- `password` specifies that user must enter a password, which will be encrypted.
- `dn` specifies that the value is a LDAP Distinguish Name.
- `email` specifies that the value is an email address.
- `url` specifies that the value is a URL address.
- `numeric` specifies that the value is a number.
- `percent` specifies that the value is a percentage.
- `number` specifies that the value is a number.
- `decimal_number` specifies that the value is a number with a decimal point.
- `number_range` specifies that the value is a range of numbers.
- `decimal_range` specifies that the value is a range of numbers that might include a decimal figure.

NOTE If creating policy, note that the policy schema only supports boolean, string, password, dn, email, numeric, percent, number, decimal_number, and number_range. It does not support paragraph, encrypted_password, decimal_range, xml, and date (some of which are not defined above).

uitype Attribute

This attribute specifies the HTML element that will be displayed in the Access Manager console. Possible values include `radio`, `link`, `button`, or `name_value_list`. No value defined for this attribute displays a default element based on the information in Table 3-1 on page 65 of Chapter 3, "The Access Manager Console."

NOTE The "type Attribute"specifies the kind of value an attribute will take. The "syntax Attribute" defines the format of that value. The "uitype Attribute" specifies the HTML element. The values of these attributes can be mixed and matched to alter the console display. See "To Change The Default Attribute Display Elements" on page 64 of Chapter 3, "The Access Manager Console," in this manual for information on how these attributes work together.

DefaultValues Sub-Element. Defining a syntax might also necessitate defining a value for the DefaultValue sub-element. A default value will then be displayed in the Access Manager console; this default value can be changed for each organization when creating a new template for the service.

CAUTION Default values of User attributes are not inherited by users when the service is assigned using the Access Manager console.

For example, all instances of the LDAP Authentication Service use the `port` attribute so a default value of 389 is defined in the XML service file. Once registered, this value can be modified for each organization using the Access Manager console. (The default value is also used by integrated applications when a service template has not been registered to an organization.) Code Example 8-8 on page 207 from `amAuthLDAP.xml` illustrates this scenario.

Code Example 8-8 DefaultValues In `amAuthLDAP.xml`

```
...
<Organization>
                <AttributeSchema name="iplanet-am-auth-ldap-server"
                    type="list"
                    syntax="string"
                    i18nKey="a101">
                    <DefaultValues>
                        <Value>identity_server_host.com:389</Value>
                    </DefaultValues>
                </AttributeSchema>
...
```

cosQualifier Attribute

This attribute defines how Access Manager resolves conflicting dynamic attribute values assigned to the same user object. The value of `cosQualifier` will appear as a qualifier to the `cosAttribute` in the LDAP entry of the CoS definition.

NOTE The priority level is defined by the Conflict Resolution Level attribute. More information on this attribute can be found in the *Sun Java System Access Manager Administration Guide*.

The value of `cosQualifier` can be defined as:

- `default` indicates that if there are two conflicting attributes assigned to the same user object, the one with the highest priority takes precedence. For more information on CoS conflicts, see Appendix E, "Directory Server Concepts," in this manual.
- `override` indicates that the CoS template value defined at the service itself overrides any priority value defined in the user entry; that is, CoS takes precedence over a defined user entry value.
- `merge-schemes` indicates that if there are two CoS templates assigned to the same user, then they are merged so that the values are combined and the user gets an aggregation of the CoS template values. For example , if there are multiple templates for a particular service that contains dynamic attributes and they are applied to a user (based on the user's roles), a merged list of attributes will be returned. `merge-schemes` works only for dynamic (or COS) type attributes.

If the `cosQualifier` attribute is not defined, the default behavior is for the user entry value to override the CoS value in the organization or role. The default value is `default`. (The `operational` value is reserved for future use.)

any Attribute

The `any` attribute specifies whether the attribute for which it is defined will display in the Access Manager console. It has six possible values that can be multiply defined using the "|" (pipe) construct:

- `display` specifies that the attribute will display on both the administrator and end user profile pages. The attribute is read / write for administrators and end users. The attribute will display on the Create page with an asterisk signifying it as a required field.

- `adminDisplay` specifies that the attribute will display on the administrator profile page only. It will not appear on the end user page; the attribute is read / write for administrators only.
- `userReadOnly` specifies that the attribute is read / write for administrators but is read only for end users. It is displayed on the end user profile pages as a non-editable label.
- `required` specifies that a value for the attribute is required in order for the object to be created. The attribute will display on the Create page with an asterisk signifying it as a required field.
- `optional` specifies that a value for the attribute is not required in order for the object to be created. The attribute will display on the Create page without an asterisk signifying it as an optional field.
- `filter` specifies that the attribute will display on the Advanced Search page.

The `required` or `optional` keywords and the `filter` and `display` keyword can be specified with a pipe symbol separating the options (`any=required|display` or `any=optional|display|filter`). If the `any` attribute is set to `display`, the qualified attribute will display in Access Manager console when the properties for the Create page are displayed. If the `any` attribute is set to `required`, an asterisk will display in that attribute's field, thus the administrator or user is required to enter a value for the object to be created in Access Manager console. If the `any` attribute is set to `optional`, it will display on the Create page, but users are not required to enter a value in order for the object to be created. If the `any` attribute is set to `filter`, the qualified attribute will display as a criteria attribute when Search is clicked from the User page.

NOTE Setting the *any* attribute to `""` (`any=""`) will prevent the attribute to which it refers from being displayed in the console.

The amAdmin.dtd Structure

The `amAdmin.dtd` defines the data structure for an XML file which can be used to perform batch operations on the directory tree using the `amAdmin` command line tool. The file reflects the structure of the Access Manager SDK and is located in the *IdentityServer_base*`/SUNWam/dtd` directory. Possible command line operations include reads and gets on the attributes as well as creations and deletions of Access Manager objects (roles, organizations, users, people containers, and groups).

NOTE XML files that are created based on the `amAdmin.dtd` are used as input with the `amAdmin` command line tool. More information on this tool can be found in the *Sun Java System Access Manager Administration Guide.*

The following sections explain the elements and attributes of the `amAdmin.dtd` using the sample XML templates installed with Access Manager for illustration. These samples can be found in *IdentityServer_base*/`SUNWam/samples/admin/cli/bulk-ops`.

Requests Element

The *Requests* element is the root element of the XML file. It must contain at least one sub-element to define the object(s) (Organization, Container, People Container, Role and/or Group, et. al.) upon which the configured actions can be performed. The *Requests* element must contain at least one of the following sub-elements:

- `OrganizationRequests`
- `SchemaRequests`
- `ServiceConfigurationRequests`
- `ContainerRequests`
- `PeopleContainerRequests`
- `RoleRequests`
- `GroupRequests`
- `UserRequests`
- `ListAccts`

To enable batch processing, the root element can take more than one of these sub-element requests.

CAUTION If multiple sub-elements are specified, they must occur in the order in which they appear in the `amAdmin.dtd`. For example, a `CreateUser` cannot come before a `CreateRole` in the same OrganizationRequests element.

Code Example 8-9 illustrates the *Requests* element tag and its corresponding *OrganizationRequests* sub-element which details the creation of two roles, two groups, a suborganization, a container, and a people container in the organization with the LDAP Distinguished Name (DN), `dc=example,dc=com`.

Code Example 8-9 Portion Of `createRequests.xml`

```
...
<Requests>
<OrganizationRequests DN="dc=example,dc=com">

    <CreateSubOrganization createDN="SubOrg1"/>
    <CreateContainer createDN="Container1"/>
    <CreatePeopleContainer createDN="People2"/>
    <CreateRole createDN="ManagerRole"/>
    <CreateRole createDN="EmployeeRole"/>
    <CreateGroup createDN="ContractorsGroup"/>
    <CreateGroup createDN="EmployeesGroup"/>

</OrganizationRequests>
...
```

OrganizationRequests Element

The *OrganizationRequests* element defines actions that can be performed on Organization objects. The required XML attribute for this element is the LDAP DN of the organization on which the configured requests will be performed. This element can have one or more sub-elements. (Different *OrganizationRequests* elements can be defined in one file to modify more than one organization.) Code Example 8-9 defines a myriad of objects to be created under the top level organization, dc=example,dc=com. The sub-elements of *OrganizationRequests* include:

- `CreateSubOrganization`
- `CreateContainer`
- `CreatePeopleContainer`
- `CreateGroupContainer`
- `CreateRole`
- `CreateUser`
- `CreateGroup`
- `CreatePolicy`
- `RegisterServices`
- `ModifySubOrganization`
- `ModifyServiceTemplate`

- `AddServiceTemplateAttributeValues`
- `RemoveServiceTemplateAttributeValues`
- `GetServiceTemplate`
- `DeleteServiceTemplate`
- `ModifyPeopleContainer`
- `ModifyRole`
- `GetSubOrganizations`
- `GetPeopleContainers`
- `GetRoles`
- `GetGroups`
- `GetUsers`
- `CreateServiceTemplate`
- `UnregisterServices`
- `GetRegisteredServiceNames`
- `GetNumberOfServices`
- `DeleteRoles`
- `DeleteGroups`
- `DeletePolicy`
- `DeletePeopleContainers`
- `DeleteSubOrganizations`
- `AddSubConfiguration`
- `DeleteSubConfiguration`
- `CreateAuthenticationDomain`
- `CreateHostedProvider`
- `CreateRemoteProvider`
- `DeleteAuthenticationDomain`
- `DeleteProvider`

- `GetProvider`
- `GetAuthenticationDomain`
- `ModifyHostedProvider`
- `ModifyRemoteProvider`
- `ModifyAuthenticationDomain`

ContainerRequests Element

The *ContainerRequests* element defines actions that can be performed on Container objects. The required XML attribute for this element is the LDAP DN of the container on which the configured requests will be performed. This element can have one or more sub-elements. (Different *ContainerRequests* elements can be defined in one file to modify more than one container.) The syntax for this element is basically the same as that of the *OrganizationRequests* element illustrated in Code Example 8-9 on page 211. The sub-elements of *ContainerRequests* can include:

- `CreateSubContainer`
- `CreatePeopleContainer`
- `CreateGroupContainer`
- `CreateRole`
- `CreateGroup`
- `CreateServiceTemplate`
- `ModifyServiceTemplate`
- `AddServiceTemplateAttributeValues`
- `RemoveServiceTemplateAttributeValues`
- `GetServiceTemplate`
- `ModifySubContainer`
- `ModifyPeopleContainer`
- `ModifyRole`
- `GetSubContainers`
- `GetPeopleContainers`
- `GetRoles`

- `GetGroups`
- `GetUsers`
- `CreateUser`
- `RegisterServices`
- `UnregisterServices`
- `DeleteServiceTemplate`
- `GetRegisteredServiceNames`
- `GetNumberOfServices`
- `DeleteRoles`
- `DeleteGroups`
- `DeletePeopleContainers`
- `DeleteSubContainers`

PeopleContainerRequests Element

The *PeopleContainerRequests* element defines actions that can be performed on People Container objects. The required XML attribute for this element is the LDAP `DN` of the people container on which the configured requests will be performed. This element can have one or more sub-elements. (Different *PeopleContainerRequests* elements can be defined in one document to modify more than one people container.) The syntax for this element is basically the same as that of the *OrganizationRequests* element illustrated in Code Example 8-9 on page 211. The sub-elements of *PeopleContainerRequests* can include:

- `CreateSubPeopleContainer`
- `ModifyPeopleContainer`
- `CreateUser`
- `ModifyUser`
- `GetNumberOfUsers`
- `GetUsers`
- `GetSubPeopleContainers`
- `DeleteUsers`
- `DeleteSubPeopleContainers`

RoleRequests Element

The *RoleRequests* element defines actions that can be performed on roles. The required XML attribute for this element is the LDAP `DN` of the role on which the configured requests will be performed. This element can have one or more sub-elements. (Different *RoleRequests* elements can be defined in one document to modify more than one role.) The syntax for this element is the same as that of the *OrganizationRequests* element illustrated in Code Example 8-9 on page 211. The sub-elements of *RoleRequests* can include:

- `CreateServiceTemplate`
- `ModifyServiceTemplate`
- `GetServiceTemplate`
- `GetNumberOfUsers`
- `GetUsers`
- `RemoveUsers`
- `AddUsers`

GroupRequests Element

The *GroupRequests* element defines actions that can be performed on Group objects. The required XML attribute for this element is the LDAP `DN` of the group on which the configured requests will be performed. This element can have one or more sub-elements. (Different *GroupRequests* elements can be defined in one document to modify more than one group.) The syntax for this element is the same as that of the *OrganizationRequests* element illustrated in Code Example 8-9 on page 211. The sub-elements of *GroupRequests* can include:

- `CreateSubGroup`
- `GetSubGroups`
- `GetNumberOfUsers`
- `GetUsers`
- `ModifySubGroups`
- `AddUsers`
- `RemoveUsers`
- `DeleteSubGroups`

UserRequests Element

The *UserRequests* element defines actions that can be performed on User objects. The required XML attribute for this element is the LDAP `DN` of the user on which the configured requests will be performed. This element can have one or more sub-elements. (Different *UserRequests* elements can be defined in one document to modify more than one user.) The syntax for this element is the same as that of the *OrganizationRequests* element illustrated in Code Example 8-9 on page 211. The sub-elements of *UserRequests* can include:

- `RegisterServices`
- `UnregisterServices`

ServiceConfigurationRequests Element

The *ServiceConfigurationRequests* element defines actions that can be performed on a specific service. The required XML attribute for this element is *serviceName*; it specifies the service on which the configured requests will be performed. This element can have one or more sub-elements. The syntax for this element is the same as that of the *OrganizationRequests* element illustrated in Code Example 8-9 on page 211. The sub-elements of *ServiceConfigurationRequests* can include:

- `AddSubConfiguration`
- `DeleteSubConfiguration`
- `DeleteAllServiceConfiguration`

AddSubConfiguration Element

The *AddSubConfiguration* element adds a secondary schema to an existing service. The AttributeValuePair Element must be defined for each attribute configured in the subconfiguration. The required XML attributes are `subConfigName`, `subConfigID`, `priority` and `serviceName`.

NOTE Attributes defined for a subconfiguration are validated against attributes defined in a subschema based on sms.dtd. A subconfiguration is defined for an organization, choosing from attributes globally defined in the subschema. For more information, see "SubSchema Element."

DeleteSubConfiguration Element

The *DeleteSubConfiguration* element deletes an existing secondary schema from a service. The required XML attributes are `subConfigName` and `serviceName` which takes a string value.

DeleteAllServiceConfiguration Element

The *DeleteAllServiceConfiguration* element deletes all configurations relating to a service and removes them from the data store. The required XML attribute is `userAtt` which specifies whether to delete the user attributes related to the service.

AttributeValuePair Element

The *AttributeValuePair* element can be a sub-element of many of the batch processing requests. It can have two sub-elements, *Attribute* and *Value*, neither of which may have sub-elements. Code Example 8-10 illustrates that a sub-people container, ou=`SubPeople2,`ou=People2,dc=example,dc=com, and a user, `dpUser`, will be created with the attributes of the two objects defined as per the attribute/value pairs.

Attribute Element

The *Attribute* element must be paired with a *Value* element. The *Attribute* element itself contains no other elements. The required XML service attribute for the *Attribute* element is `name` which is equal to the name of the attribute that is being processed. Any string format can be used without spaces.

Value Element

The *Value* element defines the value of the Attribute element. More than one *Value* element can be specified for an Attribute. The *Value* element contains no other elements and it contains no XML service attributes.

Code Example 8-10 Another Portion Of `createRequests.xml`

```
...
<PeopleContainerRequests DN="ou=People2,dc=example,dc=com">

        <CreateSubPeopleContainer createDN="SubPeople2">
                <AttributeValuePair>
                        <Attribute name="description"/>
                        <Value>SubPeople description</Value>
                </AttributeValuePair>
        </CreateSubPeopleContainer>

        <CreateUser createDN="dpUser">
           <AttributeValuePair>
              <Attribute name="cn"/>
              <Value>dpUser</Value>
           </AttributeValuePair>
           <AttributeValuePair>
              <Attribute name="sn"/>
              <Value>dpUser </Value>
           </AttributeValuePair>
           <AttributeValuePair>
              <Attribute name="userPassword"/>
```

Code Example 8-10 Another Portion Of `createRequests.xml` *(Continued)*

```
                <Value>12345678</Value>
            </AttributeValuePair>
        </CreateUser>
...
```

Create*Object* Elements

The *CreateSubOrganization, CreateContainer, CreatePeopleContainer, CreateRole, CreateGroup, CreateServiceTemplate, CreateUser, CreateSubContainer, CreateSubGroup, CreateSubPeopleContainer* elements create a sub-organization, container, people container, role, group, service template, user, sub-container, sub-group, and sub-people container, respectively. The object is created in the DN that is defined in the *<Object>Requests* element under which the particular Create*<Object>* element is being defined. AttributeValuePair Elements may be defined (or not). The required XML attribute for each element is `createDN`; it takes the `DN` of the object to be created. Code Example 8-10 on page 217 illustrates an example of *CreateSubPeopleContainer* and *CreateUser*. The DN is defined in the *PeopleContainerRequests* element as `ou=People2,dc=example,dc=com`.

NOTE *CreateGroup*/*CreateSubGroup* and *CreateRole* each have an additional attribute: `groupType` and `roleType`, respectively. `groupType` defines whether it is a static group, a filtered group or an assignable (dynamic) group. `roleType` defines whether it is a static role or a filtered role.

CreatePolicy Element

The *CreatePolicy* element creates one or more policy attributes. It takes the *Policy* element as a sub-element; `createDN` is the required XML attribute which takes the `DN` of the organization where the policy will be created. This and the following nested elements are illustrated in Code Example 8-11 on page 221. This file is `SamplePolicy.xml`, part of the policy sample application located in *IdentityServer_base*`/SUNWam/samples/policy.`

NOTE The following policy elements are the elements extracted from `amAdmin.dtd` for inclusion into the `policy.dtd`. More information can be found in the Access Manager *Administration Guide*.

Policy Element. The *Policy* sub-element defines the permissions or *rules* of the policy and to whom / what the rule applies or the *subject*. It also defines whether or not the policy is a *referral* (delegated) policy and whether there are any restrictions (or *conditions*) to the policy. It may contain one or more of the following sub-elements: *Rule, Conditions*, *Subjects*, or *Referrals*. The required XML attributes are `name` which specifies the name of the policy and `referralPolicy` which identifies whether or not the policy is a delegated one.

Rule Element. The *Rule* sub-element defines the specific permission of the policy and can take three sub-elements. The required XML attribute is `name` which defines a name for the rule. The three sub-elements are:

- ServiceName Element

  The *ServiceName* element defines the name of the service to which the policy applies. This element represents the service type. It contains no other elements. The value is exactly as that defined in the service's XML file (based on the `sms.dtd`). The XML service attribute for the *ServiceName* element is the name of the service (which takes a string value).

- ResourceName Element

  The *ResourceName* element defines the object that will be acted upon. The policy has been specifically configured to protect this object. It contains no other elements. The XML service attribute for the *ResourceName* element is the name of the object. Examples of a *ResourceName* might be `http://www.sunone.com:8080/images` on a web server or `ldap://sunone.com:389/dc=iplanet,dc=com` on a directory server. A more specific resource might be `salary://uid=jsmith,ou=people,dc=iplanet,dc=com` where the object being acted upon is the salary information of John Smith.

- AttributeValuePair Element

  The *AttributeValuePair* sub-element defines the action names and corresponding action values of the rule. For additional information, see "AttributeValuePair Element" on page 217.

Subjects Element. The *Subjects* sub-element identifies a collection of objects to which the policy applies; this overview collection is chosen based on membership in a group, ownership of a role or individual users. It takes the *Subject* sub-element. The XML attributes it can be defined with are `name` which defines a name for the collection, `description` which takes a description and `includeType` which defines whether the collection is as defined or its inverse (For example: the policy applies to users who are NOT members of the subject).

Subject Element. The *Subject* sub-element identifies a collection of objects to which the policy applies; this collection pinpoints more specific objects from the collection defined by the Subjects element. Membership can be based on roles, group membership or simply a listing of individual users. It takes as a sub-element the AttributeValuePair Element. Its required XML attribute is `type` which identifies a generic collection of objects from which the specifically defined subjects are taken. Other XML attributes include `name` which defines a name for the collection and `includeType` which defines whether the collection is as defined or its inverse (For example: the policy applies to users who are NOT members of the subject).

Referrals Element. The *Referrals* sub-element identifies a collection of policy assignments. It takes the *Referral* sub-element. The XML attributes it can be defined with are `name` which defines a name for the collection and `description` which takes a description. (Code Example 8-11 is not an example of a referral policy so there is not a Referrals element definition.)

Referral Element. The *Referral* sub-element identifies a specific policy assignment. It takes as a sub-element the AttributeValuePair Element. Its required XML attribute is `type` which identifies a generic collection of assignments from which the specifically defined referrals are taken. It can also include the `name` attribute which defines a name for the collection. (Code Example 8-11 is not an example of a referral policy so there are no Referral elements definition.)

Conditions Element. The *Conditions* sub-element identifies a collection of policy restrictions (time range, authentication level, et.al.). It must contain one or more of the *Condition* sub-element. The XML attributes it can be defined with are `name` which defines a name for the collection and `description` which takes a description.

Condition Element. The *Condition* sub-element identifies a specific policy restriction (time range, authentication level, et.al.). It takes as a sub-element the AttributeValuePair Element. Its required XML attribute is `type` which identifies a generic collection of restrictions from which the specifically defined conditions are taken. It can also include the `name` attribute which defines a name for the collection.

NOTE The Condition element might be used to configure policy for different URIs on the same domain. For example, `http://org.example.com/hr` can only be accessed by `org.example.net` from 9 am to 5 pm yet `http://org.example.com/finance` can be accessed by `org.example2.net` from 5 am to 11 pm. By defining an IP Condition attribute/value pair together with a SimpleTime Condition attribute/value pair and specifying `http://org.example.com/hr/*.jsp` as the resource, the policy would apply to all the JSPs under `http://org.example.com/hr`.

Code Example 8-11 SamplePolicy.xml

```
<Requests>
<OrganizationRequests DN="dc=iplanet,dc=com">

<CreatePolicy createDN="dc=iplanet,dc=com">
    <Policy name="PolicyOne" referralPolicy="false" >
    <Rule name="dsdasd">
    <ServiceName name="SampleWebService" />
    <ResourceName name="http://www.sun.com/public" />
        <AttributeValuePair>
            <Attribute name="GET" />
            <Value>allow</Value>
        </AttributeValuePair>
        <AttributeValuePair>
            <Attribute name="DELETE" />
            <Value>allow</Value>
        </AttributeValuePair>
        <AttributeValuePair>
            <Attribute name="PUT" />
            <Value>allow</Value>
        </AttributeValuePair>
        <AttributeValuePair>
            <Attribute name="POST" />
            <Value>allow</Value>
        </AttributeValuePair>
    </Rule>
    <Subjects name="Subjects1" description="">
    <Subject name="subject1" type="Organization">
        <AttributeValuePair>
            <Attribute name="Values"/>
            <Value>dc=iplanet,dc=com</Value>
            <Value>o=nicp,dc=iplanet,dc=com</Value>
        </AttributeValuePair>
    </Subject>
    </Subjects>
    <Conditions name="Conditions1" description="">
    <Condition name="condition1" type="SampleCondition">
        <AttributeValuePair>
            <Attribute name="userNameLength"/><Value>5</Value>
        </AttributeValuePair>
    </Condition>
    </Conditions>
    </Policy>
</CreatePolicy>

</OrganizationRequests>
</Requests>
```

CreateServiceTemplate Element

The *CreateServiceTemplate* element creates a service template for the organization defined under the second-level *Requests* element. There are no sub-elements; the *CreateServiceTemplate* element itself must be empty. The required XML attribute is `serviceName` which takes a string value. Code Example 8-12 illustrates a User service template being registered to ou=Container1,dc=example,dc=com.

Code Example 8-12 cont`CreateServiceTemplateRequests.xml` File

```
...
<Requests>
<ContainerRequests DN="ou=Container1,dc=example,dc=com">

   <CreateServiceTemplate>
       <Service_Name>iPlanetAMUserService</Service_Name>
   </CreateServiceTemplate>

</ContainerRequests>
</Requests>
```

Delete*Object* Elements

The *DeleteSubOrganizations*, *DeletePeopleContainers*, *DeleteGroups*, *DeleteRoles*, *DeleteSubContainers*, *DeleteSubGroups*, *DeleteSubPeopleContainers*, and *DeleteUsers* elements delete a sub-organization, people container, group, role, sub-container, sub-group, sub-people container and user, respectively. The object is deleted from the DN that is defined in the *<Object>Requests* element under which the particular *Delete<Object>* element is being defined. *DeleteSubOrganizations*, *DeleteUsers*, *DeleteGroups*, *DeleteSubContainers*, *DeletePeopleContainers*, *DeleteSubGroups*, *DeleteSubPeopleContainers* and *DeleteRoles* take a sub-element `DN`; only six of the listed elements have the XML attribute *deleteRecursively*. (*DeleteUsers* and *DeleteRoles* do not have this option; they have no qualifying XML attribute.) If *deleteRecursively* is set to *false*, accidental deletion of all sub-trees can be avoided; it's default value is *false*. The `DN` sub-element takes a character value equal to the DN of the object to be deleted. Code Example 8-13 illustrates an example of some of these concepts. The DN is defined in the *OrganizationRequests* element as `dc=example,dc=com`.

Code Example 8-13 orgDeleteRequests.xml

```
...
<Requests>
<OrganizationRequests DN="dc=example,dc=com">
```

Code Example 8-13 orgDeleteRequests.xml *(Continued)*

```
    <DeleteRoles>
       <DN>cn=ManagerRole,dc=example,dc=com</DN>
       <DN>cn=EmployeeRole,dc=example,dc=com</DN>
    </DeleteRoles>

    <DeleteGroups deleteRecursively="true">
       <DN>cn=EmployeesGroup,dc=example,dc=com</DN>
       <DN>cn=ContractorsGroup,dc=example,dc=com</DN>
    </DeleteGroups>

    <DeletePeopleContainers deleteRecursively="true">
       <DN>ou=People1,dc=example,dc=com</DN>
    </DeletePeopleContainers>

    <DeleteSubOrganizations deleteRecursively="true">
       <DN>o=sun.com,dc=example,dc=com</DN>
    </DeleteSubOrganizations>

</OrganizationRequests>
</Requests>
```

DeletePolicy Element

The *DeletePolicy* element takes the sub-element *PolicyName*. The *PolicyName* element has no sub-elements; it must be empty. It has a required XML attribute *name* which takes a character value equal to the name of the policy. The *DeletePolicy* element itself takes a required XML attribute: `deleteDN.` It takes a value equal to the DN of the policy to be deleted.

DeleteServiceTemplate Element

The *DeleteServiceTemplate* element deletes the specified service template. There are no sub-elements; the *DeleteServiceTemplate* element itself must be empty. The required XML attributes are `serviceName` which takes a string value and `schemaType` which defines the attribute group (Global, Organization, Dynamic, User or Policy). Code Example 8-14 illustrates the deletion of the Membership Authentication Service from `dc=example,dc=com`.

Code Example 8-14 orgDeleteServiceTemplateRequests.xml

```
<Requests>
<OrganizationRequests DN="dc=example,dc=com">
      <DeleteServiceTemplate serviceName="iPlanetAMAuthMembershipService"
schemaType="organization"/>
```

Code Example 8-14 orgDeleteServiceTemplateRequests.xml *(Continued)*

```
</OrganizationRequests>
</Requests>
```

Modify*Object* Elements

The *ModifySubOrganization*, *ModifyPeopleContainer*, *ModifySubContainer*, *ModifyRole*, and *ModifySubGroups* elements change the specified object. AttributeValuePair Elements can be defined for the listed elements. The required XML attribute is `modifyDN` which takes the `DN` of the object to be modified. Code Example 8-15 illustrates how the people container's description can be modified.

Code Example 8-15 contModifyPeoplecontainerRequests.xml

```
<Requests>
<ContainerRequests DN="dc=sun,dc=com">

    <ModifyPeopleContainer
modifyDN="ou=Test,ou=Test1,ou=People1,dc=sun,dc=com">
      <AttributeValuePair>
         <Attribute name="Description"/>
         <Value>Sun ONE Identity Server Modify</Value>
      </AttributeValuePair>
    </ModifyPeopleContainer>

</ContainerRequests>
</Requests>
```

ModifyServiceTemplate Element

The *ModifyServiceTemplate* element changes a specified service template. AttributeValuePair Element must be defined for *ModifyServiceTemplate* to change the values. The required XML attributes are `serviceName` which takes a string value, `schemaType` which defines the attribute group (Global, Organization, Dynamic, User or Policy) and `roleTemplate`. A search level attribute can also be defined. It takes a value of either `SCOPE_ONE` or `SCOPE_SUB`. `SCOPE_ONE` will retrieve just the groups at that node level; `SCOPE_SUB` gets groups at the node level and all those underneath it.

Get*Object* Elements

The *GetSubOrganizations*, *GetUsers*, *GetSubGroups*, *GetGroups*, *GetSubContainers*, *GetRoles*, *GetPeopleContainers* and *GetSubPeopleContainers* elements get the specified object. A DN may be defined as a sub-element (or not). If none is specified, ALL of the specified objects at all levels will be returned within the organization that is defined in the *<Object>Requests* element under which the particular *Get<Object>* element is being defined. The required XML attribute for all but *GetGroups* and *GetRoles* is DNsOnly and takes a true or false value. (This attribute is explained in more detail in DNs Only Attribute.) The required XML attribute of *GetGroups* and *GetRoles* is level which takes a value of either SCOPE_ONE or SCOPE_SUB. SCOPE_ONE will retrieve just the groups at that node level; SCOPE_SUB gets groups at the node level and all those underneath it. Code Example 8-16 illustrates how these elements can be modeled. The top-level DN is defined in the *OrganizationRequests* element as o=isp.

DNs Only Attribute

For all objects using the DNsOnly attribute, the *Get* elements work as stated below:

- If the element has the required XML attribute DNsOnly set to *true* and no sub-element DN is specified, only the DNs of the objects asked for will be returned.
- If the element has the required XML attribute DNsOnly set to *false* and no sub-element DN is specified, the entire object (a DN with attribute / value pairs) will be returned.
- If sub-element DNs are specified, the entire object will always be returned whether the required XML attribute DNsOnly is set to *true* or *false*.

Code Example 8-16 Portion of Batch Processing File getRequests.xml

```
...
<Requests>

<OrganizationRequests DN="o=isp">

    <GetSubOrganizations DNsOnly="false">
         <DN>o=example1.com,o=isp</DN>
         <DN>o=example2.com,o=isp</DN>
    </GetSubOrganizations>

    <GetPeopleContainers DNsOnly="false">
         <DN>ou=People,o=example1.com,o=isp</DN>
         <DN>ou=People,o=example2.com,o=isp</DN>
    </GetPeopleContainers>

    <GetRoles level="SUB_TREE"/>

    <GetGroups level="SUB_TREE"/>
```

Code Example 8-16 Portion of Batch Processing File `getRequests.xml`

```
...
<Requests>

    <GetUsers DNsOnly="false">
        <DN>cn=puser,ou=People,o=example1.com,o=isp</DN>
    </GetUsers>

</OrganizationRequests>
...
```

Get*Service* Elements

The *GetRegisteredServiceNames* and *GetNumberOfServices* elements retrieve registered services and total number of registered services, respectively. The organization from which this information is retrieved is specified in the *OrganizationRequests* element. All three elements have no sub-elements or attributes; the elements themselves must be empty. Code Example 8-17 illustrates the *GetNumberOfServices* element.

Code Example 8-17 orgGetNumberOfServiceRequests.xml

```
<Requests>

<OrganizationRequests DN="dc=example,dc=com">

    <GetNumberOfServices/>

</OrganizationRequests>

</Requests>
```

*Action*ServiceTemplate Element

The *GetServiceTemplate* and *DeleteServiceTemplate* elements get or delete a service template for the organization defined under the *OrganizationRequests* element, respectively. There are no sub-elements; the elements themselves must be empty. The required XML attributes are `serviceName` which takes a string value and `schemaType`.

*Action*ServiceTemplateAttributeValues Element

The *AddServiceTemplateAttributeValues* and *RemoveServiceTemplateAttributeValues* elements get or delete attribute values defined in a service template for the organization defined under the *OrganizationRequests* element, respectively. AttributeValuePair Element must be defined for each attribute to be added or removed. The required XML attributes are `serviceName` which takes a string value, `roleTemplate` and `schemaType` which defines the attribute group (Global, Organization, Dynamic, User or Policy). A search level attribute can also be defined. It takes a value of either `SCOPE_ONE` or `SCOPE_SUB`. `SCOPE_ONE` will retrieve just the groups at that node level; `SCOPE_SUB` gets groups at the node level and all those underneath it.

*Action*Services Elements

The *RegisterServices* and *UnregisterServices* elements perform the requested action on the service defined in the *OrganizationRequests* element. All elements take a sub-element *Service_Name* but have no XML attribute. The *Service_Name* element takes a character value equal to the name of the service. One or more *Service_Name* sub-elements can be specified.

Service Action Caveats

- The XML service file for the service must be loaded using the command line interface `amadmin` before a service can be acted upon.
- If no *Service_Name* element is specified or, in the case of *UnregisterServices*, the service was not previously registered, the request is ignored.
- If no *Service_Name* element is specified, the request will be ignored.

Code Example 8-18 illustrates how the *RegisterServices* element is modeled.

Code Example 8-18 orgRegisterServiceRequests.xml

```
<Requests>
<OrganizationRequests DN="dc=sun,dc=com">

    <RegisterServices>
        <Service_Name>sampleMailService</Service_Name>
    </RegisterServices>

</OrganizationRequests>

</Requests>
```

SchemaRequests Element

The *SchemaRequests* element consists of all requests to be performed on the XML file that defines a particular service. It has two required XML attributes: *serviceName* takes a value equal to the name of the service where the schema lives, and *SchemaType* defines the attribute group (Global, Organization, Dynamic, User or Policy). The "i18nFileName Attribute" on page 201 or a SubSchema (which specifies the complete hierarchy of the subschema separated by a "/") can also be defined.

NOTE See "Service File Naming Conventions" on page 189 for information on how the name is defined.

This element can have one or more sub-elements. (Different *SchemaRequests* elements can be defined in one document to modify more than one service.) The sub-elements of *SchemaRequests* can include:

- `RemoveDefaultValues`
- `RemovePartialDefaultValues`
- `AddDefaultValues`
- `ModifyDefaultValues`
- `GetServiceDefaultValues`
- `AddChoiceValues`
- `RemoveChoiceValues`
- `ModifyType`
- `ModifyUIType`
- `Modifyi18nKey`
- `ModifySyntax`
- `AddPropertiesViewBean`
- `AddStartRange`
- `AddEndRange`
- `AddSubSchema`
- `AddAttributeSchema`
- `RemoveSubSchema`
- `RemoveAttributeSchema`

Code Example 8-19 illustrates the opening of the *Requests* element tag and its corresponding *SchemaRequests* sub-element. The file is adding the choice *Deleted* to the Default User Status drop-down menu in the User Service.

Code Example 8-19 `schemaAddChoiceValuesRequests.xml`

```
...
<Requests>
<SchemaRequests serviceName="iPlanetAMUserService"
        SchemaType="dynamic"
        i18nKey="">
<AddChoiceValues>
     <AttributeValuePair>
          <Attribute name="iplanet-am-user-login-status"/>
                        <Value>Active</Value>
                        <Value>Inactive</Value>
                        <Value>Deleted</Value>

      </AttributeValuePair>

</AddChoiceValues>
</SchemaRequests>
</Requests>
```

RemoveDefaultValues Element

The *RemoveDefaultValues* element removes the default values from the service specified in the parent *SchemaRequests* element. It takes a sub-element of *Attribute* that specifies the service attribute which contains the values to be removed. The *Attribute* sub-element itself must be empty; it takes no sub-element. There is no required XML attribute. The syntax for this element is the same as that illustrated in Code Example 8-20.

Code Example 8-20 RemoveDefaultValues Element Code

```
...
<Requests>
<SchemaRequests serviceName="iPlanetAMUserService"
            SchemaType="dynamic">
<RemoveDefaultValues>
     <Attribute name="preferredlanguage"/>
</RemoveDefaultValues>
</SchemaRequests>
</Requests>
```

AddDefaultValues and ModifyDefaultValues Elements

The *AddDefaultValues* and *ModifyDefaultValues* elements add or change the default values from the specified schema, respectively. They take an AttributeValuePair Element which specifies the name of the attribute and the new default value; one or more attribute / value pairs can be defined. Code Example 8-21 illustrates how the *AddDefaultValues* element can be modeled.

Code Example 8-21 AddDefaultValues Element Code

```
...
<Requests>
<SchemaRequests serviceName="iPlanetAMUserService"
             SchemaType="dynamic">
<AddDefaultValues>

     <AttributeValuePair>
               <Attribute name="iplanet-am-user-auth-modules"/>
               <Value>Cert</Value>
      </AttributeValuePair>

</AddDefaultValues>
</SchemaRequests>
</Requests>
```

GetServiceDefaultValues Element

The *GetServiceDefaultValues* element retrieves the default values from the schema specified in the parent *SchemaRequests* element. There are no sub-elements; the *GetServiceDefaultValues* element itself must be empty. There is also no required XML attribute.

Federation Management Elements

The following elements consist of requests that can be performed on Access Manager configured federations. They are:

- CreateAuthenticationDomain
- DeleteAuthenticationDomain
- GetAuthenticationDomain
- ModifyAuthenticationDomain
- CreateRemoteProvider
- CreateHostedProvider
- DeleteProvider

- GetProvider
- IDPAuthContextInfo
- SPAuthContextInfo
- AuthMethodQueryString
- ModifyRemoteProvider
- ModifyHostedProvider
- ListAccts

For more information on these elements, see the DTD file itself located in the *IdentityServer_base*/`SUNWam/dtd` directory.

XML Service Files

Access Manager uses XML files to define service attributes as well as perform batch processing operations. This section contains information on the XML files included with Access Manager and how they are used.

Default XML Service Files

Access Manager installs services to manage the configurations of its components. The attributes for these services are managed using the Access Manager console; in addition, Access Manager provides code implementations to use them. These default XML service files are based on the `sms.dtd` and are located in `etc/opt/SUNWam/config/xml`. They include:

- `amAdminConsole.xml`—Defines attributes for the Administration service.
- `amAuth.xml`—Defines attributes for the Core Authentication service.
- `amAuthAnonymous.xml`—Defines attributes for the Anonymous Authentication service.
- `amAuthCert.xml`—Defines attributes for the Certificate-based Authentication service.
- `amAuthConfig.xml`—Defines configuration attributes for the Authentication service.
- `amAuthHTTPBasic.xml`—Defines attributes for the HTTP Basic Authentication service.

- `amAuthLDAP.xml`—Defines attributes for the LDAP Authentication service.
- `amAuthMembership.xml`—Defines attributes for the Membership-based Authentication service.
- `amAuthNT.xml`—Defines attributes for the Windows-based NT Authentication service.
- `amAuthRadius.xml`—Defines attributes for the Radius Authentication service.
- `amAuthSafeWord.xml`—Defines attributes for the SafeWord Authentication service.
- `amAuthSecurID.xml`—Defines attributes for the SecurID Authentication service.
- `amAuthUnix.xml`—Defines attributes for the Unix Authentication service.
- `amAuthenticationDomainConfig.xml`—Defines attributes for the Authentication Configuration service.
- `amClientData.xml`—Defines client types for the Client Detection service.
- `amClientDetection.xml`—Defines attributes for the Client Detection service.
- `amEntrySpecific.xml`—Defines attributes for the displaying attributes on the Create, Properties and Search pages for a custom service.
- `amDSS.xml`—Defines attributes for the Certificate Security service.
- `amG11NSettings.xml`—Defines attributes for the Globalization Settings service.
- `amLogging.xml`—Defines attributes for the Logging service.
- `amNaming.xml`—Defines attributes for the Naming service.
- `amPasswordReset.xml`—Defines attributes for the Password Reset service.
- `amPlatform.xml`—Defines attributes for the Platform service.
- `amPolicy.xml`—Defines attributes for the Policy service.
- `amPolicyConfig.xml`—Defines configuration attributes for the Policy service.
- `amProviderConfig.xml`—Defines attributes for Federation Management service.
- `amSAML.xml`—Defines attributes for the SAML service.
- `amSession.xml`—Defines session attributes for single sign-on.

- amUser.xml—Defines attributes for the User service.
- amWebAgent.xml—Defines attributes for the policy agents.

Modifying A Default XML Service File

Administrators can display and manage any attribute in the Access Manager console using XML service files. The new attribute(s) would need to be added to an existing XML service file. Alternately, they can be grouped into a new service by creating a new XML service file although the simplest way to add an attribute is just to extend an existing one. For example, an administrator wants to manage the nsaccountlock attribute which will give users the option of locking the account it defines. To manage it through Access Manager, nsaccountlock must be defined in a service. One option would be to add it to the amUser.xml service, iPlanetAMUserService. This is the service that, by default, includes many common attributes from the inetOrgPerson and inetUser object classes. Following is an example of how to add the nsaccountlock attribute to the amUser.xml service file.

1. Add the code illustrated in Code Example 8-22 to the SubSchema name=User element in *IdentityServer_base*/SUNWam/config/xml/amUser.xml.

Code Example 8-22 nsaccountlock Example Attribute

```
...
<AttributeSchema name="nsaccountlock"
type="single_choice"
syntax="string"
any="filter"
isChangeableByUser="yes"
i18nKey="u13">
<ChoiceValues>
        <Value>true</Value>
        <Value>false</Value>
</ChoiceValues>
<DefaultValues>
         <Value>false</Value>
</DefaultValues>
</AttributeSchema>
...
```

2. Update the *IdentityServer_base*/SUNWam/locale/en_US/amUser.properties file with the new i18nKey tag u13 as illustrated in Code Example 8-23 (including the text to be used for display).

Code Example 8-23 User Account Locked Example i18nKey

```
...
u13=User Account Locked
...
```

3. Remove the service `ou=iPlanetAMUserService,ou=services,dc=sun,dc=com` using the command line tool `amadmin`.

   For information on the `amadmin` command line syntax, see *Sun Java System Access Manager Administration Guide*.

4. Reload the modified XML service file, `amUser.xml`, using the command line tool `amadmin`.

   For information on the `amadmin` command line syntax, see *Sun Java System Access Manager Administration Guide*.

NOTE When modifying a default XML service file, be sure to also modify the Directory Server by extending the LDAP schema, if necessary. For more information, see "Defining A Custom Service" on page 187.

Batch Processing With XML Templates

The `--data` or `-t` option of `amadmin` is used to perform batch processing via the command line. Batch processing XML templates have been installed and can be used to help an administrator to:

- Create, delete and read roles, users, organizations, groups, people containers and services.
- Get roles, people containers, and users.
- Get the number of users for groups, people containers, and roles.
- Import, register and unregister services.
- Get registered service names or the total number of registered services for an existing organization.
- Execute requests in multiple XML files.

The preferred way to perform most of these functions is to use the Access Manager console. The batch processing templates have been provided for ease of use with bulk updates although they can also be used for single configuration updates. This section provides an overview of the batch processing templates which can be modified to perform batch updates in the Directory Server.

NOTE Only XML files can be used as input for the `amadmin` tool. If an administrator wants to populate the directory tree with user objects, or perform batch reads (gets) or deletes, the necessary XML input files, based on the `amAdmin.dtd` or `sms.dtd`, must be written.

XML Templates

All of the batch processing XML templates perform operations on the DIT; they create, delete, or get attribute information on user objects. These XML templates follow the structure defined by the `amAdmin.dtd` and are located in *IdentityServer_base*`/SUNWam/samples/admin/cli/bulk-ops`. The batch processing XML templates provided with Access Manager include:

- `contCreateRoleRequests.xml`—Creates a role for a container object.
- `contCreateServiceTemplateRequests.xml`—Creates a service template for a container object.
- `contModifyPeoplecontainerRequests.xml`—Modifies a people container object.
- `contModifyRoleRequests.xml`—Modifies a role assigned to a container object.
- `contModifySubcontainerRequests.xml`—Modifies a sub-container object.
- `createRequests.xml`—Creates a multitude of objects.
- `deleteGroupRequests.xml`—Deletes the sub-group of a group container.
- `getRequests.xml`—Passes information about a multitude of objects in a specific organization.
- `orgCreateServiceTemplateRequests.xml`—Creates service templates for anorganization.
- `orgDeleteRequests.xml`—Deletes a multitude of objects under a specific organization.
- `orgDeleteServiceTemplateRequests.xml`—Deletes a service template under a specific organization.

- `orgGetNumberOfServiceRequests.xml`—Passes a listing of an organization's total number of registered services.
- `orgGetRegisteredServiceRequests.xml`—Passes a listing the names of an organization's registered services.
- `orgModifyRequests.xml`—Changes values for identity-related objects in an organization.
- `orgModifyServiceTemplateRequests.xml`—Changes values for the registered service template of an organization.
- `orgRegisterServiceRequests.xml`—Registers services for an organization.
- `orgUnRegisterServiceRequests.xml`—Unregisters services for an organization.
- `pcDeleteRequests.xml`—Deletes attributes for a people container object.
- `pcModifyUserRequests.xml`—Modifies user attributes in a people container object.
- `roleCreateServiceTemplateRequests.xml`—Creates a service template for a role.
- `roleModifyServiceTemplateRequests.xml`—Changes values for the registered service template of a role.
- `schemaAddChoiceValuesRequests.xml`—Adds a selection of values to an existing service's attribute from which the user can choose.
- `schemaAddDefaultValuesRequest.xml`—Adds a default value to an existing service's attribute.
- `schemaDeleteChoiceValueRequest.xml`—Deletes a value from an existing service's attribute choices.
- `schemaDeleteDefaultValueRequest.xml`—Deletes a default value from an existing service's attribute.
- `schemaGetServiceDefaultValueRequest.xml`—Retrieves a default value from an existing service's attribute.
- `schemaModifyDefaultValueRequest.xml`—Changes the default value of an existing service's attribute.

NOTE The final XML templates (`serviceConfigurationRequests.xml`, `serviceAddSubConfigurationRequests.xml`, and `serviceDeleteSubConfigurationRequests.xml`) follow the `sms.dtd` format and are used for service sub-configurations. One use for these can be found in "Multi-LDAP Authentication Module Configuration" on page 128 of Chapter 5, "Authentication Service," in this manual.

Modifying A Batch Processing XML Template

Any of the templates discussed above can be modified to best suit the desired operation. Choose the file that performs the request, modify the elements and attributes according to the service and use the `amadmin` executable to upload the changes to Directory Server.

NOTE Be aware that creations of roles, groups, and organizations is a time-intensive operation.

Customizing User Pages

The User profile page and what attributes it displays will vary, depending on what the service developer defines. By default, every attribute in the `amUser.xml` file that has an `i18nKey` attribute specified and the `any` attribute set to display (`any=display`) will display in the Access Manager console. Alternately, if an attribute is specified to be of type `User` in another XML service file, the Access Manager console will also display it if the service is assigned to the user. Thus, User display pages in the Access Manager console can be modified to add new attributes in either of two ways:

- The `User` attribute schema definition in the specific XML service file can be modified.
- A new User schema attribute definition can be added to the User service (the `amUser.xml` service file).

For information on modifying XML service files, see "Modifying A Default XML Service File" on page 233.

NOTE Any service can describe an attribute that is for a user only. The `amUser.xml` file is just the default placeholder for user attributes that are not tied to a particular service.

Creating Users Using A Modified Directory Server Schema

There might be a need to modify the Directory Server LDAP schema in order to create users with new object classes. The procedure follows:

1. Modify the Directory Server LDAP schema with the new object classes and attributes.

   For more information on how to do this, see the Sun Java System Directory Server documentation.

2. Write a new XML service file which contains the definitions for the new object classes and attributes.

   When writing this file, the object classes should be defined under the Global element and the attributes should be defined under the User element. More information can be found in Chapter 8, "Service Management."

3. Write a new authentication module credentials file and put it in the *IdentityServer_base*`/SUNWam/lib` directory.

   This file contains the attribute-value pairs for the internationalization keys used in the file created in Step 2. More information can be found in "Configuring The Authentication Module" on page 146 of Chapter 5, "Authentication Service," in this manual.

NOTE Alternately, the path to the module configuration properties file can be put in the classpath of the web container's JVM.

4. Load the XML service file using the `amadmin` command line interface.

   More information on this tool can be found in the *Sun Java System Access Manager Administration Guide.*

5. Register the new service to the desired organization using the Access Manager console.

   For more details about registering a new service, refer to the *Sun Java System Access Manager Administration Guide.*

6. Select the new service to create a user with the additional object classes.

   When creating new user there is an option to select the newly configured service.

Service Management SDK

The Access Manager provides a Java API for service management. These interfaces can be used by developers to register services and applications, and manage their configuration data. The interfaces and methods can be found in `com.sun.identity.sm`.

ServiceSchemaManager Class

The `ServiceSchemaManager` class in the `com.sun.identity.sm` package provides interfaces to manage a service's schema. It must implement `ServiceSchema` which represents a single schema element in the service.

Retrieve Logging Location

Code Example 8-24 uses the `ServiceSchemaManager` class to retrieve the `iplanet-am-logging-location` attribute value from the Logging Service at the following DN: `ou=iPlanetAMLoggingService,ou=services,o=isp`.

Code Example 8-24 Retrieve Logging Location Sample

```
*********
SSOTokenManager manager = SSOTokenManager.getInstance();
SSOToken token = manager.createSSOToken(new
AuthPrincipal("uid=amadmin,ou=People,dc=org,dc=com"), "11111111");
ServiceSchemaManager ssm = new ServiceSchemaManager(token,
"iPlanetAMLoggingService", "1.0");
ServiceSchema ss = ssm.getGlobalSchema();
Map p = ss.getAttributeDefaults();
************
```

Retrieve User Or Dynamic Attributes

Code Example 8-25 uses the `ServiceSchemaManager` to define the `ServiceSchema` user attributes. `AMUser.getAttributes(..)` is then called to obtain the attribute / value pairs.

Code Example 8-25 Retrieve User Or Dynamic Attributes

```
ServiceSchemaManager ssm = new ServiceSchemaManager(serviceName, token);
ServiceSchema sm = ssm.getSchema(SchemaType.USER);
if (sm != null) {
```

Code Example 8-25 Retrieve User Or Dynamic Attributes

```
     Set userAttributes = ss.getAttributeSchemaNames();
     // Since USER or DYNAMIC attributes are stored as ldap attributes you
can call..
     amUser.getAttributes(userAttributes);
}
```

Retrieve Attribute Values

Code Example 8-26 illustrates one way to retrieve attribute values from a service.

Code Example 8-26 Sample Code To Retrieve Attribute Values

```
package com.iplanet.am.samples.sdk;

import java.io.*;
import java.net.*;
import java.util.*;
import com.iplanet.sso.*;
import com.iplanet.am.sdk.*;
import com.sun.identity.authentication.internal.*;
import com.sun.identity.sm.*;
import javax.servlet.*;
import javax.servlet.http.*;


public class SampleUserOperations {

    SSOToken token = null;

/**
* This user will be used for further sample operations on the
* same object
*/
    private static AMUser contextUser = null;
    private static String passWord = null;
    private static String uid = null;
    private static String lastName = null;
    private static String firstName = null;
    String userDN = null;

    private static Map scuObjMap = new HashMap();

    public static AMStoreConnection amsc = null;
    public static SampleUserOperations suo;

//Here we will try to get the value of the organization type
//attribute "iplanet-am-auth-ldap-bind-dn" of the service
//"iPlanetAMAuthLDAPService" for the organization
//DN "dc=iplanet,dc=com".
```

Code Example 8-26 Sample Code To Retrieve Attribute Values *(Continued)*

```
    public static void main(String args[]) {
        try {
            SSOTokenManager manager = SSOTokenManager.getInstance();
//If possible create the token using the tokneid or httprequest.
            SSOToken token = manager.createSSOToken(new
AuthPrincipal("uid=amadmin,ou=People,dc=iplanet,dc=com"), "11111111");
            suo = getSampleUserOperations(token);
            amsc = new AMStoreConnection(token);
            ServiceConfigManager scm = new ServiceConfigManager(token,
"iPlanetAMAuthLDAPService", "1.0");
            String orgName = "dc=iplanet,dc=com";
            ServiceConfig sc = scm.getOrganizationConfig(orgName, null);
            Map mp = sc.getAttributes();
            Iterator itr =
((HashSet)mp.get("iplanet-am-auth-ldap-bind-dn")).iterator();
            System.out.println("bind dn for the org -" + orgName + "-is-" +
(String)itr.next());
            System.exit(0);
        } catch (Exception e) {
            System.out.println("Exception Message: " + e.getMessage());
            e.printStackTrace();
        }
    }

/* Basic Constructor */

    public SampleUserOperations(SSOToken token) {
        this.token = token;
        scuObjMap.put(token, this);
    }

/* Use the same  object for multiple operations */

    public static SampleUserOperations getSampleUserOperations(SSOToken
token) {
        SampleUserOperations scuObj =
(SampleUserOperations)scuObjMap.get(token);
        if (scuObj == null ) {
            scuObj = new SampleUserOperations(token);
        }
        return scuObj;
    }

/**
* This method will describe the SDK usage for creating a user.
* It uses AMStoreConnection to get the organization object
* It uses the Set Parameters to store the different attributes of
* the user. This method is used for command line.
* It throws an AMException if unable to create it and we throw
* message "unable to create" to the GUI by catching the same
*/
    public String createUser(AMStoreConnection conn) {
        try {
            Map userAttributeMap = new HashMap();
```

Code Example 8-26 Sample Code To Retrieve Attribute Values *(Continued)*

```
                uid = "user";
                storeUserAttributes("uid", uid, userAttributeMap);
                firstName = "user";
                storeUserAttributes("givenname", firstName,
userAttributeMap);
                lastName = "one";
                storeUserAttributes("sn", lastName, userAttributeMap);
                passWord = "userone";
                storeUserAttributes("userPassword", passWord,
userAttributeMap);

            Map userMap1 = new HashMap();
            userMap1.put(uid, userAttributeMap);
/**
* Provide the DN according to the DIT
*/
            String dn = "ou=People,o=iplanet.com,o=isp";
            AMPeopleContainer ampc = conn.getPeopleContainer(dn);
            ampc.createUsers(userMap1);
            userDN = "uid=" + uid + "," + dn;
/*
* This is to keep the context of the user
*/
            contextUser = conn.getUser(userDN);
            return "Successfully added the user: " + uid;
        } catch (Exception ex) {
            ex.printStackTrace();
        }
          return "Unable to create";
    }

/**
* This method will describe the SDK usage for creating a user.
* It uses AMStoreConnection to get the organization object
* It uses the Set Parameters to store the different attributes of
* the user.
* It throws an AMException if unable to create it and we throw
* message "unable to create" to the GUI by catching the same
*/

    public String createUser(HttpServletRequest req, Set parameters,
AMStoreConnection
conn) {
        try {
            Map userAttributeMap = new HashMap();
            if (parameters.contains("uid")) {
                uid = req.getParameter("uid");
                storeUserAttributes("uid", uid, userAttributeMap);
            }
            if(parameters.contains("firstname")) {
                firstName = req.getParameter("firstname");
                storeUserAttributes("givenname", firstName,
userAttributeMap);
            }
```

Code Example 8-26 Sample Code To Retrieve Attribute Values *(Continued)*

```
            if(parameters.contains("lastname")) {
                lastName = req.getParameter("lastname");
                storeUserAttributes("sn", lastName, userAttributeMap);
            }
            if(parameters.contains("password")) {
                passWord = req.getParameter("userPassword");
                storeUserAttributes("userPassword", passWord,
userAttributeMap);
            }

            Map userMap1 = new HashMap();
            userMap1.put(uid, userAttributeMap);
            String orgDN = req.getParameter("orgName");
            String dn = "ou=People" + "," + orgDN;
            AMPeopleContainer ampc = conn.getPeopleContainer(dn);
            ampc.createUsers(userMap1);
            userDN = "uid=" + uid + "," + dn;
            /*
             * This is to keep the context of the user
             */
            contextUser = conn.getUser(userDN);
            return showCreateUserSuccess();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
          return "Unable to create";
    }

/**
* This method describes the SDK usage for modifying the user.
*/
    public String modifyUser(HttpServletRequest req) {
        HashMap modifyMap = new HashMap();
        lastName = req.getParameter("lastname");
        storeUserAttributes("sn", lastName, modifyMap);
        firstName = req.getParameter("firstname");
        storeUserAttributes("givenname", firstName, modifyMap);
        passWord = req.getParameter("userpassword");
        storeUserAttributes("userPassword", passWord, modifyMap);

        try {
            contextUser.setAttributes(modifyMap);
            contextUser.store();
            return showModifyUserSuccess();
        } catch (Exception ex) {
            System.out.println("Exception occured");
        }
        return "Unable to modify";
    }

/**
* This method describes the SDK usage for deleting the user.
*/

    public String deleteUser() {
```

Code Example 8-26 Sample Code To Retrieve Attribute Values *(Continued)*

```
        try {
            contextUser.delete(false);
            return "Deleted successfully";
        } catch (Exception ex) {
            System.out.println("Exception occured");
        }
        return "Unable to delete";
    }

/* This method is for the GUI purposes */

    public String showCreateUser() {
        StringBuffer sb = new StringBuffer();
        sb.append("<HTML>");
        sb.append("<HEAD>");
        sb.append("</HEAD>");
        sb.append("<BODY>");
        sb.append("<FORM name=\"allattributes\" METHOD=POST
ACTION=\"/amserver/sdksample\">");
        sb.append("<TABLE>");
        sb.append("<TR>");
        sb.append("<TD ALIGN=LEFT VALIGN=MIDDLE><B>Login ID</B></TD>");
        sb.append("<TD VALIGN=MIDDLE><INPUT TYPE=\"text\" NAME=\"uid\"
VALUE=\"\"
SIZE=32 MAXLENGTH=64></TD><TD><B>Under Organization</B></TD>");
        sb.append("<TD VALIGN=MIDDLE><INPUT TYPE=\"text\" NAME=\"orgName\"
VALUE=\"\"
SIZE=32 MAXLENGTH=64></TD>");
        sb.append("</TR>");
        sb.append("<TR>");
        sb.append("<TD ALIGN=LEFT VALIGN=MIDDLE><B>First Name</B></TD>");
        sb.append("<TD VALIGN=MIDDLE><INPUT TYPE=\"text\" NAME=\"firstname\"
VALUE=\"\" SIZE=32 MAXLENGTH=64></TD>");
        sb.append("</TR>");
        sb.append("<TR>");
        sb.append("<TD ALIGN=LEFT VALIGN=MIDDLE><B>Last Name</B></TD>");
        sb.append("<TD VALIGN=MIDDLE><INPUT TYPE=\"text\" NAME=\"lastname\"
VALUE=\"\"
SIZE=32 MAXLENGTH=64></TD>");
        sb.append("</TR>");
        sb.append("<TR>");
        sb.append("<TD ALIGN=LEFT VALIGN=MIDDLE><B>Password</B></TD>");
        sb.append("<TD VALIGN=MIDDLE><INPUT TYPE=\"password\"
NAME=\"userpassword\"
VALUE=\"\" SIZE=12></TD>");
        sb.append("</TR>");
        sb.append("<TR>");
        sb.append("<TD ALIGN=LEFT VALIGN=MIDDLE><B>Confirm
Password</B></TD>");
        sb.append("<TD VALIGN=MIDDLE><INPUT TYPE=\"password\"
NAME=\"passwordagain\"
VALUE=\"\" SIZE=12></TD>");
        sb.append("</TR>");
        sb.append("<TR>");
```

Code Example 8-26 Sample Code To Retrieve Attribute Values *(Continued)*

```
        sb.append("<TD><input type=SUBMIT NAME=\"usersubmit\">");
        sb.append("</TD></TR>");
        sb.append("</TABLE>");
        sb.append("</FORM>");
        sb.append("</BODY>");
        sb.append("</HTML>");
        return sb.toString();
    }

    private void storeUserAttributes(String attribute, String value, Map
userMap) {
        Set userSet = new HashSet();
        userSet.add(value);
        userMap.put(attribute, userSet);
    }

/* This method is for the GUI purposes */

    private String showCreateUserSuccess() {
        StringBuffer sb = new StringBuffer();
        sb.append("<HTML>");
        sb.append("<HEAD>");
        sb.append("</HEAD>");
        sb.append("<BODY>");
        sb.append("Created Successfully");
        sb.append("<FORM name=\"usersuccessful\" METHOD=POST
ACTION=\"/amserver/sdksample\">");
        sb.append("<TABLE>");
        sb.append("<TR>");
        sb.append("<TD><input type=SUBMIT NAME=\"modifyuser\"
VALUE=\"Modify\">");
        sb.append("</TD></TR>");
        sb.append("</TABLE>");
        sb.append("</FORM>");
        sb.append("</BODY>");
        sb.append("</HTML>");
        return sb.toString();
    }

    /* This method is for the GUI purposes */

    public String showModifyUser() {
        StringBuffer sb = new StringBuffer();
        sb.append("<HTML>");
        sb.append("<HEAD>");
        sb.append("</HEAD>");
        sb.append("<BODY>");
        sb.append("uid:" + uid);
        sb.append("<FORM name=\"showmodify\" METHOD=POST
ACTION=\"/amserver/sdksample\">");
        sb.append("<TABLE>");
        sb.append("<TR>");
        sb.append("<TD ALIGN=LEFT VALIGN=MIDDLE><B>First Name</B></TD>");
        sb.append("<TD VALIGN=MIDDLE><INPUT TYPE=\"text\" NAME=\"firstname\"
```

Code Example 8-26 Sample Code To Retrieve Attribute Values *(Continued)*

```
VALUE=\"");
        sb.append(firstName + "\" SIZE=32 MAXLENGTH=64></TD>");
        sb.append("</TR>");
        sb.append("<TR>");
        sb.append("<TD ALIGN=LEFT VALIGN=MIDDLE><B>Last Name</B></TD>");
        sb.append("<TD VALIGN=MIDDLE><INPUT TYPE=\"text\" NAME=\"lastname\"
VALUE=\"");
        sb.append(lastName + "\" SIZE=32 MAXLENGTH=64></TD>");
        sb.append("</TR>");
        sb.append("<TR>");
        sb.append("<TD ALIGN=LEFT VALIGN=MIDDLE><B>Password</B></TD>");
        sb.append("<TD VALIGN=MIDDLE><INPUT TYPE=\"password\"
NAME=\"userpassword\"
VALUE=\"");
        sb.append(passWord + "\" SIZE=12></TD>");
        sb.append("</TR>");
        sb.append("<TR>");
        sb.append("<TD><input type=SUBMIT NAME=\"modifyusersubmit\">");
        sb.append("</TD></TR>");
        sb.append("</TABLE>");
        sb.append("</FORM>");
        sb.append("</BODY>");
        sb.append("</HTML>");
        return sb.toString();
    }

/* This method is for the GUI purposes */

    private String showModifyUserSuccess() {
        StringBuffer sb = new StringBuffer();
        sb.append("<HTML>");
        sb.append("<HEAD>");
        sb.append("</HEAD>");
        sb.append("<BODY>");
        sb.append("Modified Successfully");
        sb.append("<FORM name=\"modifyusersuccessful\" METHOD=POST
ACTION=\"/amserver/sdksample\">");
        sb.append("<TABLE>");
        sb.append("<TR>");
        sb.append("<TD><input type=SUBMIT NAME=\"deleteusersubmit\"
VALUE=\"Delete\">");
        sb.append("</TD></TR>");
        sb.append("</TABLE>");
        sb.append("</FORM>");
        sb.append("</BODY>");
        sb.append("</HTML>");
        return sb.toString();
    }

/* This method is for the GUI purposes */

    public String showDeleteUser() {
        StringBuffer sb = new StringBuffer();
        sb.append("<HTML>");
        sb.append("<HEAD>");
```

Code Example 8-26 Sample Code To Retrieve Attribute Values *(Continued)*

```
        sb.append("</HEAD>");
        sb.append("<BODY>");
        sb.append("<FORM name=\"showdelete\" METHOD=POST
ACTION=\"/amserver/sdksample\">");
        sb.append("<TABLE>");
        sb.append("<TR>");
        sb.append("<TD><input type=SUBMIT NAME=\"deleteusersubmit\">");
        sb.append("</TD></TR>");
        sb.append("</TABLE>");
        sb.append("</FORM>");
        sb.append("</BODY>");
        sb.append("</HTML>");
        return sb.toString();
    }

}
```

Chapter 9

Policy Management

Sun Java™ System Access Manager 6 2005Q1 includes a Policy Management feature that allows you to define, manage, and enforce policies that control access to protected resources. It allows administrators to configure and administer these conditions for applications, resources, and identities managed within the Access Manager deployment. This chapter explains the Policy Management feature and its architecture. It contains the following sections:

- "Policy SDK" on page 249
- "Extending the Policy Management Feature" on page 257

Policy SDK

The Policy SDK provides Java and C APIs to allow external applications to participate in its functionality. With the SDK, applications can determine privileges and manage policies.

The *Sun Java™ System Access Manager Developer's Reference* provides summaries of data types, structures, and functions that make up the public Access Manager C APIs. You will find the Javadoc for Access Manager Java APIs in this location:

IdentityServer_base/`SUNWam/docs/am_public_javadocs.jar`

Java SDK For Policy

The crux of the Policy Service is the Java SDK. It defines the following packages:

- `com.sun.identity.policy` provides the APIs for administering (creating, deleting, modifying) and evaluating policies. It is used by the Access Manager console and/or the command line interface.

- `com.sun.identity.policy.interfaces` provides source interfaces used to implement custom subjects, conditions, referrals and resource comparators.
- `com.sun.identity.policy.client` are APIs used by remote Java applications that need to evaluate policies and get policy decisions.

TIP `AMConfig.properties` must be copied from Access Manager to a client machine as well as the respective jars to run test code in a remote environment. Some properties (like the notification url for remote client) need to be modified for their functionality to work.

Policy API For Java

The `com.sun.identity.policy` package provides the classes and methods to manage, administer and evaluate policies. They can be used by the Access Manager console or the `amadmin` command line interface tool. Select classes and methods are discussed in this section.

Policy Evaluation Classes

The following information introduces some of the classes that can be used to evaluate configured policies for access to a protected resource.

PolicyEvaluator Class `com.sun.identity.policy.PolicyEvaluator` can be integrated into Java applications to evaluate policy privileges and provide policy decisions. This class provides support for both boolean and non-boolean type policies. A `PolicyEvaluator` is created by calling the constructor with a service name. Public methods of this class include:

- `isAllowed`—evaluates the policy associated with the given resource and returns a boolean value indicating whether the policy evaluation resulted in an allow or deny.
  - Returns a boolean value of:
    - `true` if access is allowed.
    - `false` if access is denied.

NOTE A boolean false value overrides a boolean true value. Once an action is determined to have a false value, other values are not evaluated.

  - Arguments:

        - `com.iplanet.sso.SSOToken`: The SSOToken associated with the principal for which the policy will be evaluated.
        - `java.lang.String resourceName`: A string representing the requested resource.
        - `java.lang.String actionName`: The action for which the policy will be evaluated. In a typical web application scenario, the action could be `GET` or `POST`.
        - `java.util.Map envParameters`: A map containing environment parameters that may be needed to successfully evaluate the associated policies.
    - Exceptions:
        - Throws `com.iplanet.sso.SSOException` if the given session token is not valid or has expired.
        - Throws `com.sun.identity.policy.PolicyException` if the result could not be computed for any reason other than a token problem.
- `getPolicyDecision`—evaluates the policy and ascertains privileges for non-boolean decisions. It returns a decision that gives a user permission to perform a specific action on a specific resource. This method can also check permissions for multiple actions.
    - Returns `com.sun.identity.policy.PolicyDecision`.
    - Arguments:
        - `com.iplanet.sso.SSOToken`: The SSO token associated with the principal for which the policy will be evaluated.
        - `java.lang.String resourceName`: A string representing the requested resource.
        - `java.util.Set actionName`: A collection of actions for which the policy will be evaluated.
        - `java.util.Map envParameters`: A map containing environment parameters that may be needed to successfully evaluate the associated policies.
    - Exceptions:
        - Throws `com.iplanet.sso.SSOException` if the given session token is not valid or expired.

    - Throws `com.sun.identity.policy.PolicyException` if the result could not be computed for any reason other than a token problem.

- `getResourceResult`—obtains the policy and ascertains privileges for non-boolean decisions. Possible values for the scope of this method are `self` and `subtree`. `self` gets the policy decision for the specified resource only. `subtree` includes the policy decisions for all resources (defined in the policies) which are sub-resources of the specified resource.

To illustrate, the `PolicyEvaluator` class can be used to display the links for a list of resources to which an authenticated user has access. The `getResourceResult` method would be used to get the list of resources. The `resourceName` parameter would be `http://host.domain:port` which would return all the resources to which the user has access on that server. These resources are returned as a `PolicyDecision` based on the user's defined policies. If the user is allowed to access resources on different servers, this method needs to be called for each server.

NOTE Not all resources that have policy decisions are accessible to the user. The `ActionDecision`(s) contained in policy decisions carry this information.

ProxyPolicyEvaluator Class

`com.sun.identity.policy.ProxyPolicyEvaluator` allows a privileged user (top level administrator, organization administrator, policy administrator, or organization policy administrator) to get policy privileges and evaluate policy decisions for any user in their respective scope of administration. `com.sun.identity.policy.ProxyPolicyEvaluatorFactory` is the singleton class used to get `ProxyPolicyEvaluator` instances.

Code Example 9-1 Public Methods For ProxyPolicyEvaluator

```
/**
 * Evaluates a simple privilege of boolean type. The privilege
 * indicates if the user identified by the principalName
 * can perform specified action on the specified resource.
 *
 * @param principalName principal name for whom to
 * compute the privilege.
 * @param resourceName name of the resource
 * for which to compute policy result.
 * @param actionName name of the action the user is trying to
 * perform on the resource
 * @param env run time environment parameters
 *
 * @return the result of the evaluation as a boolean value
 *
 * @throws PolicyException exception form policy framework
```

Code Example 9-1 Public Methods For ProxyPolicyEvaluator *(Continued)*

```
     * @throws SSOException if sso token is invalid
     *
     */
    public boolean isAllowed(String principalName, String resourceName,
        String actionName, Map env) throws PolicyException, SSOException;

    /**
     * Gets policy decision for the user identified by the
     * principalName for the given resource
     *
     * @param principalName principal name for whom to compute the
     * policy decision
     * @param resourceName name of the resource for which to
     * compute policy decision
     * @param env run time environment parameters
     *
     * @return the policy decision for the principal for the given
     * resource
     * @throws PolicyException exception form policy framework
     * @throws SSOException if sso token is invalid
     *
     */
    public PolicyDecision getPolicyDecision(String principalName,
        String resourceName, Map env)
        throws PolicyException, SSOException;

    /**
     * Gets protected resources for a user identified by the
     * principalName. Conditions defined in the policies
     * are ignored while computing protected resources.
     * Only resources that are  subresources of the  given
     * rootResource or equal to the  given rootResource would
     * be returned.
     * If all policies applicable to a resource are
     * only referral policies, no ProtectedResource would be
     * returned for such a resource.
     * @param principalName principal name for whom
     * to compute the privilege.
     * @param rootResource  only resources that are subresources
     * of the given rootResource or equal to the  given
     * rootResource would be returned. If
     * <code>PolicyEvaluator.ALL_RESOURCES</code>
     * is passed as rootResource, resources under
     * all root  resources of the service
     * type are considered while computing protected
     * resources.
     *
     * @return set of protected resources. The set contains
     * ProtectedResource objects.
     *
     * @throws PolicyException exception form policy framework
     * @throws SSOException if sso token is invalid
     * @see ProtectedResource
     *
     */
```

Code Example 9-1 Public Methods For ProxyPolicyEvaluator *(Continued)*

```
    public Set getProtectedResourcesIgnoreConditions(String principalName,
        String rootResource)  throws PolicyException, SSOException
```

PolicyEvaluator Class `com.sun.identity.policy.client.PolicyEvaluator` evaluates policies and provides policy decisions for remote applications which do not have a direct access to Directory Server (for example, if there is a firewall). The `com.sun.identity.policy.client.PolicyEvaluator` defined in "PolicyEvaluator Class" on page 250 requires direct LDAP access to policies stored in Directory Server. This class `com.sun.identity.policy.client.PolicyEvaluator` is implemented using XML over HTTP(s). It stores a cache of policy decisions for faster responses and maintains the cache in sync with the Policy Service on the instance of Access Manager using the notification and polling mechanism.

NOTE The PolicyEvaluator class can be used in a deployment container running Access Manager, or in a stand alone Java Virtual Machine (JVM) running the Access Manager SDK. Respective to the JVM, a property must be defined to point to `serverconfig.xml` which, in turn, points to Directory Server. This is done by launching the JVM with the following argument:

```
-D
"com.iplanet.coreservices.configpath=/etc/opt/SUNWam/
config/ums"
```

Policy Management Classes

The following classes can be used by system administrators to manage policies in Access Manager. The interfaces for this functionality are also found in the `com.sun.identity.policy` package.

PolicyManager `com.sun.identity.policy.PolicyManager` is the top level administrator class for policy management, providing methods that allow an administrator to create, modify or delete an organization's policies. The `PolicyManager` can be obtained by passing a privileged user's session token or by passing a privileged user's session token with an organization name. Some of this class's more widely used methods include:

- `getPolicyNames` - retrieves all named policies created for the organization for which the policy manager was instantiated. This method can also take a pattern (filter) as an argument.
- `getPolicy` - retrieves a policy when given the policy's name.

- `addPolicy` - adds a policy to the specified organization. If a policy with the same name already exists, it will be overwritten.
- `removePolicy` - removes a policy from the specified organization.

Policy `com.sun.identity.policy.Policy` represents a policy definition with all its intended parts (rules, subjects, referrals and conditions). The policy object is saved in the data store only when the `store` method is called or if the `addPolicy` or `replacePolicy` methods from the `PolicyManager` class are invoked. This class contains methods to add, remove, replace or get any of the parts of a policy definition.

PolicyEvent `com.sun.identity.policy.PolicyEvent` represents a happening in a policy that could potentially change the current access status. For example, a policy event would be created and passed to the registered policy listeners whenever there is a change in a policy rule. This class works with the `PolicyListener` class in the `com.sun.identity.policy.interface` package.

Policy Plugin API For Java

The following classes are used by service developers and policy administrators who need to provide additional policy features as well as support for legacy policies. The package for these classes is `com.sun.identity.policy.interfaces`. The interfaces include:

ResourceName

`ResourceName` provides methods to determine the hierarchy of the resource names for a determined service type. For example, these methods can check to see if two resources names are the same or if one is a sub-resource of the other.

Subject

`Subject` defines methods that can determine if an authenticated user (possessing an SSOToken) is a member of the given subject.

Referral

`Referral` defines methods used to delegate the policy definition or evaluation of a selected resource (and its sub-resources) to another organization or policy server.

Condition

`Condition` provides methods used to constrain a policy; for example, time of day or IP address. This interface allows the pluggable implementation of the conditions.

PolicyListener

`PolicyListener` defines an interface to register for policy events when a policy is added, removed or changed. It is used by the policy service to send notifications and by listeners to review policy change events.

C Library For Policy

Access Manager also provides a library of policy evaluation APIs to enable integration of the policy functionality into for C applications. The C library provides a comprehensive set of interfaces that query policy results of an authenticated user for a given action on a given resource. The result of the policy evaluation is called an *action value* and may not always be binary (allow / deny or yes / no); action values can also be non-boolean. For example, John Smith has a mailbox quota of 100MB. 100 is the value defined by a policy. As policy evaluation results in string values only, the policy evaluation returned is 100 numeric not 100MB. It is up to the application developer to define metrics for the values obtained appropriately.

CAUTION Previous releases of Access Manager contained C libraries in *IdentityServer_base*/`lib/capi`. The `capi` directory is being deprecated, and is currently available for backward compatibility. It will be removed in the next release, and therefore it is highly recommended that existing application paths to this directory are changed and new applications do not access it. Paths include `RPATH`, `LD_LIBRARY_PATH`, `PATH`, compiler options, etc.)

As the first step of policy implementation, the API abstracts how a resource is represented by mandating that any resource be represented in a string format. For example, on a web server, resources may be represented as URLs. The policy evaluation engine cares only about the relative relevance of one resource to other. There are five relative relevances defined between two resources, namely: *exact match*, *no match*, *subordinate match*, *superior match* or *exact pattern match*. Having represented the resources in string format, the service developer must provide interfaces that establish the relevant relationship between resources.

NOTE *Exact pattern match* is a special case where resources may be represented collectively as patterns. The information is abstracted from the policy service and the comparison operation must take a boolean parameter to trigger a pattern matched comparison. During the caching of policy information, the policy engine does not care about patterns, whereas during policy evaluation, the comparisons are pattern sensitive.

The service developer must also provide a method to extract the root of the given resource. For example, in a URL, the `protocol://`*identity_server_host.domain_name*`:`*port* portion represents the root. The three functions (`has_patterns`, `get_resource_root` and `compare_urls`) are specializations of resource representations. The set of characteristics needed to define a resource is called a *resource trait*. Resource traits are taken as a parameter during service initialization in the `am_resource_traits_t` structure. Using the resource traits, the policy service constructs a resource graph for policy evaluation. In a web server policy sense, the relation between all the resources in the system spans out like a tree with the `protocol://`*identity_server_host.domain_name*`:`*port*`/` being the root of the tree.

NOTE The policy management system is generic and makes no assumptions about any particular policy definition requirement.

Policy Evaluation API for C

Two opaque data structures are defined: `am_map_t` and `am_properties_t`. `am_map_t` provides a key to multiple value mapping and `am_properties_t` provides a key to single value mapping. `am_properties_t` provides the additional functionality of loading a configuration file and getting values of specific data types. These are simple data structures that are only used for information exchange to and from the policy evaluation interfaces.

Extending the Policy Management Feature

Out of the box, Access Manager provides the URL Policy Agent service for policy enforcement. However, you can use the Policy API to extend the functionality of the default policy service. Through the API, you can create a new policy service to fit your needs.

Access Manager provides a collection of sample files to illustrate how to use the Policy API. This section explains how to use the samples to develop and add custom subjects, conditions and referrals to existing policy, to programatically construct new policies, and to develop and run policy evaluation programs.In order to successfully execute the policy samples, the following tasks must be completed in order:

1. Compiling the Policy Samples
2. Adding the Policy Service to Access Manager

3. Developing Custom Subjects, Conditions and Referrals
4. Creating Policies for the Service
5. Developing and Running Policy Evaluation Programs

The samples and all associated files are located in the following directories:

IdentityServer_base/`SUNWam/samples/policy` (Solaris)

IdentityServer_base/`identity/samples/policy` (Linux)

NOTE Throughout the rest of this chapter, only the Solaris directory information will be given. Please note that the directory structure for Linux is different.For more information, please see "Conventions Used in This Guide" on page 30.

Compiling the Policy Samples

Before you can use the files included with the samples, you must compile them. To compile the samples:

1. Update the following variables in the `Makefile`:

   `BASE` - Set this variable to refer *IdentityServer_base*/`SUNWam`.

   `JAVA_HOME` - Set this variable to your installation location of JDK. The JDK version should be higher than JDK 1.3.1.

   `CLASSPATH` - Set this variable to refer to all of the `jar` files

2. Compile the samples by running `gmake all`.

Adding the Policy Service to Access Manager

Before you use the API to customize the interface, you must add the `SampleWebService.xml` file to Access Manager. For information on adding new policy services, see the "Policy Management" chapter of the Access Manager *Administration Guide.*

Developing Custom Subjects, Conditions and Referrals

The Policy API provides a means to customize a policy service interface, which provides the variables that define the policy itself. This sample shows how to customize the subject, condition and rule interfaces for `SampleWebService.`

The interfaces used to implement the customization are as follows:

- `SampleSubject.java` - Implements the Subject interface. This subject applies to all authenticated users who have valid SSOTokens.
- `SampleConditon.java` - Implements the Condition interface. This condition makes the policy applicable to users whose name length is grater or equal to the length spcified in the condition.
- `SampleReferral.java` - Implements the Referral interface. This referral retrieves the referral policy decision from the SampleReferral.properties file. This file is located in the same directory as the rest of the sample files.

The subject, condition and referral implementations need to be added to `iPlanetAMPolicyServicea` and `iPlanetAMPolicyConfigService` services in order to make them available for policy definitions. (These services are loaded into Access Manager during installation.) To add the sample implementations to the policy framework, you must first modify the `iPlanetAMPolicy` service and `iPlanetAMPolicyConfig` service. The policy samples provide a modified XML file for use with each service. The `iPlanetAMPolicyServicea` service uses `amPolicy.xml` and the `iPlanetAMPolicyConfigService` uses `amPolicyConfig.xml.`

The following XML attribute values in `amPolicyConfig.xml` must be changed to reflect your installation before they are loaded to Access Manager:

- `iplanet-am-policy-config-ldap-server`
- `iplanet-am-policy-config-ldap-base-dn`
- `iplanet-am-policy-config-ldap-bind-dn`
- `iplanet-am-policy-config-ldap-bind-password.`

When setting the `iplanet-am-policy-config-ldap-bind-password` attribute, the encrypted value must be used. The `ampassword` command can be used to generate encrypted password (for more information, see “The ampassword Command Line Tool” in the Access Manager Administration Guide”). Alternatively, they can be set to correct values when the policy configuration service is registered for the organizations.

To Load the Modified Services

1. Back up `iPlanetAMPolicy` and `iPlanetAMPolicyConfig` services using the `db2ldif` utility. For example:

   ```
   cd DirectoryServer_base/slapd-hostname

   db2ldif -n userRoot -s
   "ou=iPlanetAMPolicyService,ou=services,root_suffix"

   db2ldif -n userRoot -s
   "ou=iPlanetAMPolicyConfigService,ou=services,root_suffix"
   ```

2. Remove the existing `iPlanetAMPolicy` and `iPlanetAMPolicyConfig` services by running the following commands:

   ```
   IdentityServer_base/SUNWam/bin/amadmin

       --runasdn "uid=amAdmin,ou=People,default_org,root_suffix"

       --password password

       --deleteservice iPlanetAMPolicyService

   IdentityServer_base/SUNWam/bin/amadmin

       --runasdn "uid=amAdmin,ou=People,<default_org>,root_suffix"

       --password password

       --deleteservice  iPlanetAMPolicyConfigService
   ```

3. Add the modified services back to the server. The XML attributes values must be modified to your installation before running these commands):

   ```
   IdentityServer_base/SUNWam/bin/amadmin

       --runasdn "uid=amAdmin,ou=People,default_org,root_suffix"

       --password password

       --schema IdentityServer_base/SUNWam/samples/policy/amPolicy.xml


   IdentityServer_base/SUNWam/bin/amadmin

       --runasdn "uid=amAdmin,ou=People,default_org,root_suffix"

       --password password

       --schema
   IdentityServer_base/SUNWam/samples/policy/amPolicyConfig.xml
   ```

   The original services XML files for these two services are located in *IdentityServer_base*`/SUNWam/config/xml`.

4. Change the properties files with the following commands:

```
cd IdentityServer_base/SUNWam/locale
mv amPolicy.properties amPolicy.properties.bak
mv amPolicy_en.properties amPolicy_en.properties.bak
mv amPolicyConfig.properties amPolicyConfig.properties.bak
mv amPolicyConfig_en.properties amPolicyConfig_en.properties.bak
cp IdentityServer_base/SUNWam/samples/policy/amPolicy.properties
cp IdentityServer_base/SUNWam/samples/policy/amPolicy_en.properties
cp IdentityServer_base/SUNWam/samples/policy/amPolicyConfig.properties
cp
IdentityServer_base/SUNWam/samples/policy/amPolicyConfig_en.properties
```

5. To deploy the sample plugins copy `SampleSubject.class`, `SampleCondition.class` and `SampleReferral.class` from the sample directory to *IdentityServer_base*/`SUNWam/lib`.

6. Restart Access Manager.

7. Login into Access Manager console and register policy configuration service to the organization. (For more information, see the “Policy Management” chapter of the Access Manager Administration Guide.)

   You can also use `amadmin` tool to register policy configuration service to organizations.

8. Enter the LDAP Bind password for the LDAP Bind User.

   The sample subject, condition and referral implementations are now available for policy management through the Access Manager console or the `amadmin` tool.

Creating Policies for the Service

After you add the `SampleWebService` service to Access Manager and develop the custom interfaces, you need to create a policy for the service. Access Manager provides the following sample policy definitions for the `SampleWebService`:

- SamplePolicy.xml - Defines a normal policy.
- SamplereferralPolicy.xml - Defines a referral policy.

For information on adding new policy services, see the “Policy Management” chapter of the Access Manager *Administration Guide*.

Developing and Running Policy Evaluation Programs

The Policy API provides a Policy Evaluation API that allows you to write a policy evaluation program to ensure that the policy service, and the policy definitions that the service contains, function properly.

The Policy Evaluation API has one java class, `PolicyEvaluator`, and the package for this class is `com.sun.identity.policy.PolicyEvaluator`. Based on this class, Access Manager provides a sample policy evaluation program called `PolicyEvaluation.java`.

The sample policy evaluation program uses the `PolicyEvaluation.properties` file, in which you specify the input for the evaluation program such as service name, action names, condition environment parameters, user name, user password and so forth. The following properties can be set as input to the evaluation program:

- Set the value of `pe.servicename` to the service name (`SampleWebService`).
- Set the `pe.resoucename` to the resource name against which you want to evaluate the policy.
- Specify the action names in the `pe.actionnames`. Separate the action names with ':'. If you want to get all the action values, you can simply leave the `pe.actionnames` blank.
- Set other required properties like `pe.username`, `pe.password.`
- Set the optional properties `pe.authlevel`, `pe.authscheme`, `pe.requestip`, `pe.dnsname`, `pe.time` if you use the corresponding conditions in your policy definitions.

NOTE Before you run the policy evaluation program, make sure that you have set up the policy definitions.

To Run the Policy Evaluation Program

1. Set the environment variable `LD_LIBRARY_PATH` to `/usr/lib/mps/secv1`.

2. Run the evaluation sample program, use the `gmake` command.

   The policy decision from the policy evaluation program is displayed on the terminal.

Constructing Policies Programmatically

The Policy API provides Policy Management API that allows you to programatically create, add, update and remove policies. Access Manager provides a sample program, `PolicyCreator.java`, which demonstrates how to construct policies and add them to the policy store. For your reference, the `PolicyCreator.java` code is listed at the end of this section.

In this sample, the following two policies are created:

- `policy1`- Normal policy, which contains one subject of each subject type and one condition of each condition type that are provided by Access Manager out of box
- `refpolicy1`- Referral policy.

To Run PolicyCreator.java

1. Compile sample Java programs. See "Compiling the Policy Samples" on page 258 for more information.
2. Set the environment variable `LD_LIBRARY_PATH` to `/usr/lib/mps/secv1`.

In the Access Manager console, create a suborganization called `org1`, a user called `user1`, a group called `group1` and role called `role1`. Make sure that all of these identity objects are created in your top-level organization. For more information on creating these objects, see the Access Manager *Administration Guide*.

3. Set the values of following properties in the `PolicyEvaluation.properties` file:

   - `pe.orgname` - DN of the top level organazation.
   - `pe.username` - userid to authenticate.
   - `pe.password` - password to use to authenticate.

4. Use the following command to create the policies:

   ```
   gmake createPolicies
   ```

5. In the Access Manager console, verify that `policy1` and `refpolicy1` were added.

PolicyCreator.java

The following section lists the `PolicyCreator.java` code.

Code Example 9-2 PolicyCreator.java

```
import com.sun.identity.policy.PolicyManager;
import com.sun.identity.policy.ReferralTypeManager;
import com.sun.identity.policy.SubjectTypeManager;
import com.sun.identity.policy.ConditionTypeManager;
import com.sun.identity.policy.Policy;
import com.sun.identity.policy.Rule;
import com.sun.identity.policy.interfaces.Referral;
import com.sun.identity.policy.interfaces.Subject;
import com.sun.identity.policy.interfaces.Condition;
import com.sun.identity.policy.PolicyException;

import com.iplanet.sso.SSOToken;
import com.iplanet.sso.SSOException;

import java.util.Set;
import java.util.HashSet;
import java.util.Map;
import java.util.HashMap;

public class PolicyCreator {

    public static final String DNS_NAME="DnsName";
    public static final String DNS_VALUE="*.red.iplanet.com";
    public static final String START_TIME="StartTime";
    public static final String START_TIME_VALUE="08:00";
    public static final String END_TIME="EndTime";
    public static final String END_TIME_VALUE="21:00";
    public static final String AUTH_LEVEL="AuthLevel";
    public static final String AUTH_LEVEL_VALUE="0";
    public static final String AUTH_SCHEME="AuthScheme";
    public static final String AUTH_SCHEME_VALUE="LDAP";


    private String orgDN;
    private SSOToken ssoToken;
    private PolicyManager pm;

    private PolicyCreator() throws PolicyException, SSOException {
        BaseUtils.loadProperties();
        orgDN = BaseUtils.getProperty("pe.orgname");
        System.out.println("orgDN = " + orgDN);
        ssoToken = BaseUtils.getToken();
        pm = new PolicyManager(ssoToken, orgDN);
    }

    public static void main(String[] args) {
        try {
            PolicyCreator pc = new PolicyCreator();
            pc.addReferralPolicy();
import com.sun.identity.policy.PolicyManager;
            pc.addNormalPolicy();
            System.exit(0);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    private void addNormalPolicy() throws PolicyException, SSOException
{
        System.out.println("Creating normal policy in org:" + orgDN);
        PolicyManager pm = new PolicyManager(ssoToken, orgDN);
        SubjectTypeManager stm = pm.getSubjectTypeManager();
        ConditionTypeManager ctm = pm.getConditionTypeManager();

        Policy policy = new Policy("policy1", "policy1 description");
        Map actions = new HashMap(1);
        Set values = new HashSet(1);
        values.add("allow");
        actions.put("GET", values);
        String resourceName = "http://myhost.com:80/hello.html";
        Rule rule = new Rule("rule1", "iPlanetAMWebAgentService",
                resourceName, actions);
        policy.addRule(rule);

        Subject subject = stm.getSubject("Organization");
        Set subjectValues = new HashSet(1);
        subjectValues.add(orgDN);
        subject.setValues(subjectValues);
        policy.addSubject("organization", subject);

        subject = stm.getSubject("LDAPUsers");
        subjectValues = new HashSet(1);
        String userDN = "uid=user1,ou=people" + "," + orgDN;
        subjectValues.add(userDN);
        subject.setValues(subjectValues);
        policy.addSubject("ldapusers", subject);

        subject = stm.getSubject("LDAPGroups");
        subjectValues = new HashSet(1);
        String groupDN = "cn=group1,ou=groups" + "," + orgDN;
        subjectValues.add(groupDN);
        subject.setValues(subjectValues);
        policy.addSubject("ldapgroups", subject);

        subject = stm.getSubject("LDAPRoles");
        subjectValues = new HashSet(1);
        String roleDN = "cn=role1" + "," + orgDN;
        subjectValues.add(roleDN);
        subject.setValues(subjectValues);
        policy.addSubject("ldaproles", subject);

        subject = stm.getSubject("IdentityServerRoles");
        subjectValues = new HashSet(1);
```

```
import com.sun.identity.policy.PolicyManager;
        roleDN = "cn=role1" + "," + orgDN;
        subjectValues.add(roleDN);
        subject.setValues(subjectValues);
        policy.addSubject("is-roles", subject);

        Condition condition = ctm.getCondition("IPCondition");
        Map conditionProperties = new HashMap(1);
        Set propertyValues = new HashSet(1);
        propertyValues.add(DNS_VALUE);
        conditionProperties.put(DNS_NAME, propertyValues);
        condition.setProperties(conditionProperties);
        policy.addCondition("ip_condition", condition);

        condition = ctm.getCondition("SimpleTimeCondition");
        conditionProperties = new HashMap(1);
        propertyValues = new HashSet(1);
        propertyValues.add(START_TIME_VALUE);
        conditionProperties.put(START_TIME, propertyValues);
        propertyValues = new HashSet(1);
        propertyValues.add(END_TIME_VALUE);
        conditionProperties.put(END_TIME, propertyValues);
        condition.setProperties(conditionProperties);
        policy.addCondition("time_condition", condition);

        condition = ctm.getCondition("AuthLevelCondition");
        conditionProperties = new HashMap(1);
        propertyValues = new HashSet(1);
        propertyValues.add(AUTH_LEVEL_VALUE);
        conditionProperties.put(AUTH_LEVEL, propertyValues);
        condition.setProperties(conditionProperties);
        policy.addCondition("auth_level_condition", condition);


        condition = ctm.getCondition("AuthSchemeCondition");
        conditionProperties = new HashMap(1);
        propertyValues = new HashSet(1);
        propertyValues.add(AUTH_SCHEME_VALUE);
        conditionProperties.put(AUTH_SCHEME, propertyValues);
        condition.setProperties(conditionProperties);
        policy.addCondition("auth_scheme_condition", condition);


        pm.addPolicy(policy);

        System.out.println("Created  normal policy");
    }

    private void addReferralPolicy()
            throws PolicyException, SSOException {
        System.out.println("Creating referral policy for org1");
        ReferralTypeManager rtm = pm.getReferralTypeManager();
        String subOrgDN = "o=org1" +  "," + orgDN;
        Policy policy = new Policy("refpolicy1", "ref to org1" true);
```

```
import com.sun.identity.policy.PolicyManager;
        Map actions = new HashMap(1);
        Rule rule = new Rule("rule1",
"iPlanetAMWebAgentService","http://myhost.com:80/org1", actions);
        policy.addRule(rule);
        Referral referral = rtm.getReferral("SubOrgReferral");
        Set referralValues = new HashSet(1);
        referralValues.add(subOrgDN);
        referral.setValues(referralValues);
        policy.addReferral("ref to org1" , referral);
        pm.addPolicy(policy);
        System.out.println("Created referral policy for org1");
    }

}
```

Chapter 10

Using the JAAS Authorization Framework

Previous versions of Access Manager (Identity Server 6.0 and 6.1) provide custom policy APIs to define and evaluate access policies. This model provided centralized management of policies in its own policy store, the Sun ONE or Java Enterprise System (JES) Directory Server. In Sun Java™ System Access Manager 6 2005Q1 and beyond, the authorization segment of the Java Authentication and Authorization Service (JAAS) framework is added to the original model. This new model is based on JAAS 1.0 and Java 2 Platform, Standard Edition (J2SE) 1.3.1.

Access Manager now bridges the gap between J2SE and Access Manager APIs. In this new framework, Access Manager maps its private APIs to JAAS interfaces. This makes it possible for you to use the JAAS interface to access the Access Manager policy framework.

The topics covered in this chapter are:

- "Overview of JAAS Authorization" on page 269
- "JAAS Authorization in Access Manager" on page 274
- "Enabling the JAAS Authorization Framework" on page 276

Overview of JAAS Authorization

JAAS is a set of APIs that enable services to authenticate and enforce access controls upon users. It implements a Java technology version of the standard Pluggable Authentication Module (PAM) framework, and supports user-based authorization. JAAS authorization extends the Java security architecture which uses a security policy to specify what access rights are granted to executing code. That architecture, introduced in the Java 2 platform, is code-based. The permissions are granted based on code characteristics such as where the code is coming from, whether it is digitally signed, and if so, the identity of the signer.

Code Example 10-1 illustrates a Java security policy. This grants the code in the am_services.jar file, located in the current directory, the specified permission. No signer is specified, so it doesn't matter whether the code is signed or not.

Code Example 10-1 Example of a Java Security Policy

```
grant codebase Cfile:./am_services.jar" {
            permission javax.security.auth.AuthPermission
                             "createLoginContext.AMLoginContext";
         };
```

JAAS authorization adds user centric access control that applies control based on what code is running as well as on who is running it.

By default, JAAS comes with a reference implementation of Policy (`com.sun.security.auth.PolicyFile`) which is file-based. This implementation parses the Java.policy file `${java.home}/lib/`*security directory* and uses that to direct the associations of permissions to code. You can change the pointer to some other `PolicyFile` implementation or use a combination of files. By default, two files are consulted to evaluate policy. One is `com.sun.security.auth.PolicyFile`, mentioned above, and the other is `.java.policy` as defined in user's home directory.

To make JAAS authorization take place, include a Principal field in the grant statement or statements in your policy file. A Principal field indicates which user executing the code is allowed the designated permissions. The Policy file grant statements can now optionally include one or more Principal fields. Including Principal field in the grant statement indicates that the user represented by the specified Principal, who is executing the specified code, has the designated permissions. See the Principal field example in Code Example 10-2.

Code Example 10-2 A Policy File Grant Statement

```
                grant   codebase "file:./am_services.jar",
                       Principal javax.security.auth.XXXprincipal
                            "your_user_name@your_domain"  {

                  permission java.util.PropertyPermission "java.home", "read";
```

Code Example 10-2 A Policy File Grant Statement

```
                  permission java.util.PropertyPermission "user.home", "read";
                   permission java.io.FilePermission "foo.txt", "read";
                };
```

How Policy Enforcement Works

The Java 2 runtime enforces access controls via the `java.lang.SecurityManager`, which is consulted any time untrusted code attempts to perform a sensitive operation (accesses to the local file system, for example). To determine whether the code has sufficient permissions, the SecurityManager implementation delegates responsibility to the `java.security.AccessController`, which first obtains an image of the current AccessControlContext, and then ensures that the retrieved AccessControlContext contains sufficient permissions for the operation to be permitted.

JAAS supplements this architecture by providing the method `Subject.doAs` to dynamically associate an authenticated subject with the current `AccessControlContext`. As subsequent access control checks are made, the `AccessController` can base its decisions upon both the executing code itself, and upon the principals associated with the subject. Access Manager provides support for JAAS authentication, which results in the population of the subject with Principals that represents the user.

Code Example 10-3 The `Subject.doAs` Method

```
public final class Subject {
                ...
                // associate the subject with the current
                // AccessControlContext and execute the action
                public static Object doAs(Subject s,
                                java.security.PrivilegedAction action) { }
            }
```

To illustrate a usage scenario for the `doAs` method, consider a service that authenticates a remote subject, and then performs some work on behalf of that subject. For security reasons, the server should run in an `AccessControlContext` bound by the subject's permissions. Using JAAS, the server can ensure this by preparing the work to be performed as a `java.security.PrivilegedAction`. Then, by invoking the doAs method, the server provides both the authenticated subject and the prepared `PrivilegedAction`. The `doAs` implementation associates the subject with the current AccessControlContext and then executes the action. When security checks occur during execution, the Java 2 SecurityManager queries the JAAS policy, updates the current AccessControlContext with the permissions granted to the subject and the executing codesource, and then performs its regular permission checks. When the action is completed, the doAs method removes the subject from the current AccessControlContext, and returns the result back to the caller. Code Example 10-4 illustrates this flow.

Code Example 10-4 Sample Code for Subject.doAS

```
public static void main(String[] args) {
        try {
            // Create an SSOToken
            AuthContext ac = new AuthContext("dc=iplanet,dc=com");
            ac.login();
            Callback[] callbacks = null;
            if (ac.hasMoreRequirements()) {
                callbacks = ac.getRequirements();

                if (callbacks != null) {
                    try {
                        addLoginCallbackMessage(callbacks); // this method
                      sets appropriate responses in the callbacks.
                        ac.submitRequirements(callbacks);
                    } catch (Exception e) { }
                }
            }
            if (ac.getStatus() == AuthContext.Status.SUCCESS) {
                Subject subject = ac.getSubject(); // get the authenticated
                     subject
                  FilePermission perm = new FilePermission("/tmp/test",
                    "read");
                  Subject.doAs(subject, new PrivilegedExceptionAction() {
                      /* above statement means execute  run() method of the
                     Class PrivilegedExceptionAction()
                          as the specified subject */
                      public Object run() throws IOException  {
                          // if the above run()  was not throwing Exception
                         could have created an instance of PrivilegedAction
                          // instead of  PrivilegedExceptionAction here
                          AccessController.checkPermission(perm);
                            File = new File("/tmp/test");
                            return null;
                      }
                 });
             }
   }
```

In this example, the `AccessController` is checking the application's current policy implementation. If any permission defined in the policy file implies the requested permission, the method will simply return; otherwise an AccessControlException will be thrown. The check is actually redundant in this example, because the constructor for the default File implementation performs the same check. This samples is meant to illustrate the flow.

How the JS2E Access Controller Works

`AccessController` works with the `java.security.Policy` implementation to securely process application requests. In JS2E, a typical `checkPermission(Permission p)` method call on the `AccessController` class might result in the following sequence:

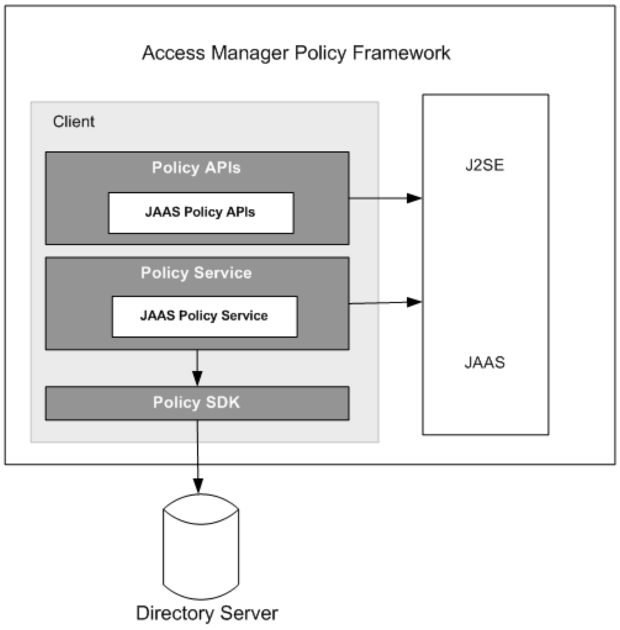
1. The `AccessController` invokes the `getPermisisons()` method of the `javax.security.auth`.policy passing in the subject and the code source.
2. The `getPermissions()` method returns a `PermissionCollection` class instance, which represents a collection of same types of permissions.
3. The `elements()` method of the returned `PermissionCollection` gets called, which returns an enumeration of the permissions held in this `PermissionCollection`.
4. For each of the permissions returned in the enumeration (in step 3), the `perm.newPermissionCollection()` method gets called to obtain the `PermissionCollection` used to store the permission.
5. `PermissionCollection.add(perm)` gets called by the J2SE internal code to store the permission in its `PermissionCollection`.
6. The `AccessController` calls the `implies(Permission p)` method of the `PermissionCollection` returned in step 2.
7. Once the `implies()` of `PermissionCollection` is called, it in turn triggers the calling of `implies(Permission p)` of the individual permission objects contained in the `PermissionCollection`. These methods return `true` if the current permission object in the collection implies the specified permission; the methods return `false` the current permission object in the collection does not imply the specified permission. This outcome is implementation dependent and can be changed.

JAAS Authorization in Access Manager

Access Manager provides a custom implementation of the JAAS `javax.security.auth.Policy`. The customized implementation leverages the J2SE access controller and security manager to provide policy evaluation for all Access Manager related permissions. The customized implementation also falls back on the J2SE default Policy implementation `com.sun.security.auth.PolicyFile` for access to system level resources. Access Manager policy does not control access to `com.sun.security.auth.PolicyFile`.

Figure 10-1 illustrates how the JAAS framework works within the Access Manager policy framework.

Figure 10-1 JAAS Authorization Framework



Access Manager uses both JAAS and J2SE's file-based policy for all the resources for which Access Manager does not provide access control. For Access Manager resources such as URLs and so forth, new policy and permissions are defined. This model leverages the best of JAAS and the best of J2SE in one solution. It uses the JAAS framework for its default access control where needed, and then enhances the framework to incorporate the Access Manager policy evaluation. In this way, you can use the Access Manager policy implementation to make policy evaluations pertaining to Access Manager policies, but revert back to the default method of controlling access to resources not under Access Manager control.

Custom APIs

Access Manager provides the following custom APIs:.

- `Package com.sun.identity.policy.jaas`

  This package includes classes for performing policy evaluation against Access Manager using JAAS (Java Authentication and Authorization) framework.

- `ISPermission`

  This class provides the support for JAAS Authorization service. It is a new JAAS Permission which extends the Permission class and is defined to evaluate permission against the Access Manager policy framework.

- `ISPolicy`

  This is an implementation of abstract class `javax.security.auth.Policy` for representing the system security policy for a Java application environment. It performs policy evaluation against the Access manager policy service instead of against the default file-based `PolicyFile`.

For a comprehensive listing of related APIs, see the Javadoc in the following directory: *AccessManager-base*`/SUNWam/docs`.

User Interface

The user interface for entering permissions and policy is the Access Manager administration console which works with the policy administration API. Once the policy is defined, the evaluation is done using the J2SE architecture and enhanced policy implementation.

`ISPermission` covers the case when additional policy services are defined and imported, provided they only have boolean action values. In fact boolean evaluation is all that can be done using JAAS since JAAS permissions have a boolean result.

Enabling the JAAS Authorization Framework

You enable the JAAS authorization framework by resetting policy. Use the `Policy.setPolicy(Policy)` API to reset policy during run time. In Code Example 10-5, `Policy.setPolicy(com.sun.identity.policy.jaas.ISPolicy)` resets the policy. In this example, the client application wants to use JAAS authorization API to communicate with the Access Manger and to perform policy evaluation. Access Manager provides the support needed to use Access Manager policy so that policy can be defined through the new `ISPermission`.

Code Example 10-5 Sample JAAS Authorization Code

```
  public static void main(String[] args) {
         try {
             // Create an SSOToken

            AuthContext ac = new AuthContext("dc=iplanet,dc=com");
             ac.login();
             Callback[] callbacks = null;
             if (ac.hasMoreRequirements()) {
                 callbacks = ac.getRequirements();

                 if (callbacks != null) {
                     try {
                         addLoginCallbackMessage(callbacks); // this method
                           sets appropriate responses in the callbacks.
                         ac.submitRequirements(callbacks);
                     } catch (Exception e) { }
                 }
             }
             if (ac.getStatus() == AuthContext.Status.SUCCESS) {
                 Subject subject = ac.getSubject(); // get the authenticated
                    subject
                     Policy.setPolicy(new ISPolicy()); // change the
                     policy to  our own Policy
                      ISPermission perm = new
                              ("iPlanetAMWebAgentService",
                         "http://www.sun.com:80", "GET");
                   Subject.doAs(subject, new PrivilegedExceptionAction() {
                       /* above statement means execute  run() method of the
                         Class PrivilegedExceptionAction()
                           as the specified subject */
                       public Object run() throws Exception {
                           AccessController.checkPermission(perm);
                          // the above will return quietly if the Permission
                               has been granted else will throw access denied
                             // Exception, so if the above highlighed
                                   ISPermission had not been granted, this
                                  code would have
                                     // thrown an Exception.
                           return null;
                       }
```

Code Example 10-5 Sample JAAS Authorization Code *(Continued)*

```
                    });
                }
    }
```

Chapter 11

SAML Service

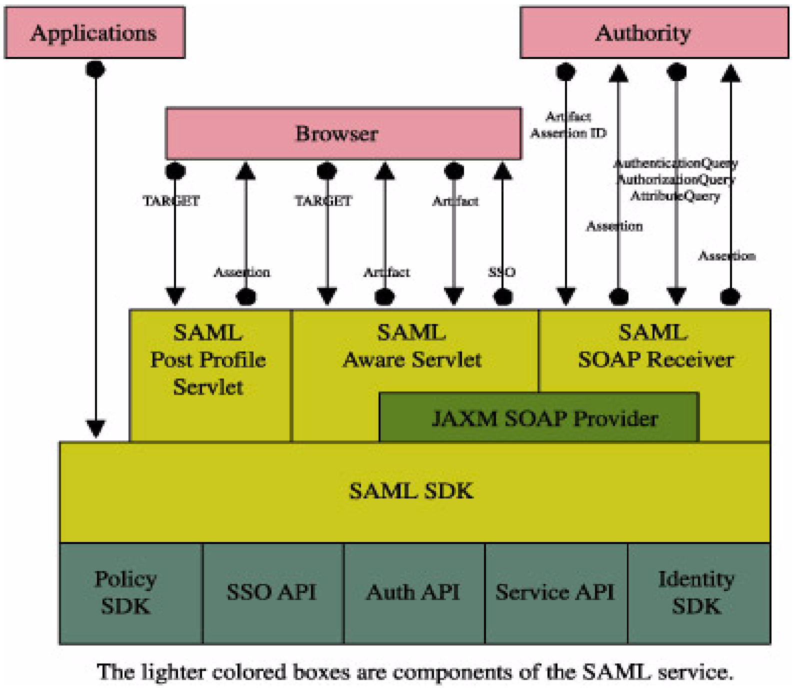
Sun Java™ System Access Manager 6 2005Q1 uses the Security Assertion Markup Language (SAML) for exchanging security information. SAML defines an eXtensible Markup Language (XML) framework to achieve inter-operability across different vendor platforms that provide SAML assertions. This chapter explains SAML and defines how it is used within Access Manager. It contains the following sections:

Overview

SAML is an open-standard protocol that uses an XML framework to exchange security information between an authority and a trusted partner site. The security information concerns itself with authentication status, access authorization decisions and subject attributes. The Organization for the Advancement of Structured Information Standards (OASIS) drives the development of the SAML specifications. The latest SAML information and specifications can be found at the Oasis Security Services Technical Committee home page.

SAML security information is expressed in the form of an assertion about a subject. A *subject* is an entity in a particular domain, either human or machine, with which the security information concerns itself. (A person identified by an email address is a subject as might be a printer.) An *assertion* is a package of verified security information that supplies one or more statements concerning a subject's authentication status, access authorization decisions or attributes. Assertions are issued by a SAML authority. (An *authority* is a platform or application that has been integrated with the SAML SDK, allowing it to relay security information.) The assertions are received by partner sites defined within the authority as *trusted*. SAML authorities use different sources to configure the assertion information including external data stores or assertions that have already been received and verified. Figure 11-1 illustrates how the SAML Service interacts with the other Access Manager components.

Figure 11-1 SAML Interaction Within Access Manager



The SAML Service allows Access Manager to work in the following ways:

- Users can authenticate against Access Manager and access trusted partner sites without having to reauthenticate. (This is a single sign-on process independent of the proprietary Access Manager process discussed in Chapter 4, "Single Sign-On And Sessions," of this manual.)
- Access Manager acts 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.

- Access Manager acts as both an attribute authority (allowing trusted partner sites to query a subject’s attributes) and an authentication authority (allowing trusted partner sites to query a subject’s authentication information.)
- Two parties in different security domains can validate each other for the purpose of performing business transactions.
- The SAML SDK can be used to build Authentication, Authorization Decision and Attribute Assertions.
- The SAML Service provides pluggable XML-based digital signature signing and verifying.

NOTE Although the Federation Management module integrates aspects of the SAML specifications, it is independent of the Access Manager SAML Service as described in this chapter.

Accessing The SAML Service

The SAML Service can be accessed using a web browser or the SAML SDK. An end user would authenticate to Access Manager using a web browser and, when authorized to do so, access URLs from trusted partner sites. Developers, on the other hand, would integrate the API into their applications to enable them to exchange security information with Access Manager. For example, a Java application can use the SAML API to accomplish single sign-on. After obtaining a `SSOToken` from Access Manager, the application can call the `doWebArtifact()` method of the `SAMLClient` class which will send a SOAP request for authorization information to Access Manager and, if applicable, redirect the application to the destination site.

SAML Component Details

The following sections explain specific details of the components of the SAML Service. They include:

- Profile Types
- Assertion Types
- SAML SOAP Receiver

Profile Types

A set of rules describing how to embed and extract SAML assertions is called a *profile*. The profile describes how the assertions can be combined with other objects by an authority, transported from the authority and, subsequently, processed at the trusted partner site. Access Manager supports two profiles that use HTTP: the Web Browser Artifact Profile and the Web Browser POST profile. Either of these profiles can be used in the case of single sign-on between two SAML-enabled entities, allowing an already authenticated user to access resources from a trusted partner site. Each profile has its benefits that include:

- Because Web Browser POST profile does not require the SOAP, it is more firewall-friendly and involves less steps and server side processing.
- Web Browser Artifact Profile requires less processing overhead because there is no assertion signing as there is in Web Browser POST profile.
- Web Browser Artifact Profile works without Javascript-enabled browsers.

NOTE The profile methods can be initiated through a web browser or the SAML API. More information on the API method can be found in "SAML SDK" on page 289.

Web Browser Artifact Profile

The Web Browser Artifact Profile defines interaction between three parties: a user equipped with a web browser, an authority site, and a trusted partner site. When an authenticated user attempts to access a trusted partner site (generally by clicking a link), they are directed to a transfer service at the authority site. In Access Manager, the transfer service is the SAML Aware Servlet. The base of the transfer URL is `http(s)://`*identity_server_host.domain_name*`:`*port*`/`*server_deploy_uri*`/SAMLAwareServlet`; it is appended with the URL of the location to which the user is requesting access (`?TARGET=URL_of_destination`). The SAML Aware Servlet then provides the following functions as part of the Web Browser Artifact Profile:

1. It compares the SAML Service's configured list of Trusted Partner Sites against the user's TARGET location.

   Only targets configured in the Trusted Partner Sites attribute of the SAML Service can access the SAML Service. Configured targets specify a domain and/or a port number. More information on this attribute can be found in the *Sun Java System Access Manager Administration Guide.*

2. Assuming the TARGET location was found in the list of Trusted Partner Sites, the SAML Aware Servlet looks for and validates the session token from the inbound request.

   Without a valid session token, Access Manager will not create an assertion.

3. The SAML Aware Servlet then creates an *artifact* and a corresponding *assertion*.

   An *artifact* is carried as part of the URL and points to an assertion and its source; it is not, and does not contain, the security information itself. The *assertion* contains the security information and is built from the user's session information and optional attribute information from the `siteAttributeMapper` class. (More information on the `siteAttributeMapper` can be found in "com.sun.identity.saml.plugins" on page 291.) The assertion can be signed.

NOTE The need to send an artifact rather than the assertion itself is dictated by the restrictions on URL size imposed by many web browsers.

4. It redirects the user's browser to the Artifact Receiver URL with a query string containing the artifact and the original TARGET location.

   The Artifact Receiver URL is based on mapping configurations defined in the SAML Service. More information on this can be found in the SAML Service Attributes chapter of the *Sun Java System Access Manager Administration Guide*.

NOTE In Access Manager, the Artifact Receiver URL and the SAML Aware Servlet are one and the same. Other SAML implementations might not integrate the two servlets.

5. At the Artifact Receiver URL, the artifact is extracted from the query string to find the SOAP Receiver URL.

   The SAML SDK extracts the source ID from the artifact and uses it to find the SOAP Receiver URL in the SAML Service configuration. "SAML SOAP Receiver" on page 286 has more information on the use of SOAP, a communications specification integrating XML and HTTPS.

6. A SAML request containing the artifact is then sent to the SOAP Receiver URL at the trusted partner site requesting the assertion to which the artifact points.

   The Artifact Receiver URL uses SOAP binding to request the assertion.

7. The SOAP Receiver URL accepts the returned artifact query from the trusted partner site and responds by sending the correct assertion in a SOAP response.

8. The assertion is processed, mapping the user account information from the trusted partner site to the target site's user account.

   The user is either granted or denied access to the trusted partner site. If access is granted a `SSOToken` is generated, a cookie is set to the browser and the user is redirected to the TARGET location.

NOTE A sample has been provided to test the Web Browser Artifact Profile function. "SAML Samples" on page 295 has more information.

Web Browser POST Profile

The Web Browser POST Profile allows security information to be supplied to a trusted partner site using the HTTP POST method (and without the use of an artifact). It consists of two interactions: the first between a user with a web browser and the Access Manager, and the second between the same user and a trusted partner site.

When an authenticated user attempts to access a trusted partner site using a web browser (usually by clicking a link), they are redirected to a transfer service in the authority site. In Access Manager, the transfer service is the SAML Post Profile Servlet. The base of the transfer URL is `http(s)://`*identity_server_host.domain_name*`:`*port*`/`*server_deploy_uri*`/SAMLPOSTProfileServlet`; it is appended with the URL of the location to which the user is requesting access (`?TARGET=URL_of_destination`). The SAML POST Profile Servlet provides functions for the two POST Profile interactions. In the first interaction between the user and Access Manager:

1. Access Manager obtains the TARGET location from the request and retrieves the trusted partner site URL from the SAML Service.

   Again, only targets configured in the Trusted Partner Sites attribute of the SAML Service can access the SAML Service. More information on this can be found in the SAML Service Attributes chapter of the *Sun Java System Access Manager Administration Guide.*

2. It generates an assertion using the `AssertionManager` class of the SAML SDK.

   "com.sun.identity.saml" on page 289 contains information on the `AssertionManager` class.

3. It forms, signs and Base64 encodes a `SAMLResponse` containing the assertion.

4. It generates an HTML form, containing both the `SAMLResponse` and the TARGET as parameters, and posts the form as an HTTP response back to the user's browser.

5. The user's browser is then directed to the location based on this information.

In the second interaction between the user and the trusted partner site:

1. The trusted partner site obtains the TARGET and `SAMLResponse` from the request.

2. It Base64 decodes the `SAMLResponse`.

3. It verifies the signature on the `SAMLResponse` and obtains and verifies the SAML response itself.

   It also verifies the assertion inside the `SAMLResponse` and enforces single-sign on policy.

4. It obtains or creates an `SSOToken` and redirects the authenticated user to the TARGET location.

The POST profile function is provided by either of two means: an HTTP request using the `SAMLPOSTProfileServlet`, or an `SAMLClient` API call [`doWebPost( )`] to a Java application.

NOTE A sample has been provided to test the Web Browser POST Profile function. "SAML Samples" on page 295 has more information.

Single Use Policy With POST Profile

According to the SAML specifications, the trusted partner site MUST ensure a single-use policy for SSO assertions communicated by the Web POST Profile. Thus, the `SAMLPOSTProfileServlet` maintains a store of SSO assertion IDs and the time they expire. When an assertion is received, the servlet first checks for an entry in the map. If one exists, the servlet returns an error. If not, the assertion ID and expiration time is saved to the map. The `POSTCleanUpThread` removes expired assertion IDs periodically.

Assertion Types

SAML assertions are represented as XML constructs based on a schema located at `http://www.oasis-open.org/committees/security/docs/cs-sstc-schema-assertion-01.xsd`. The SAML specification provides for several types of assertions that are also defined in the SAML Service:

- An *authentication assertion* declares that the specified subject has been authenticated by a particular means at a particular time. In Access Manager, the Authentication Service is the authentication authority. Code Example 11-1 illustrates a sample authentication assertion.

Code Example 11-1 Sample Authentication Assertion

```
<?xml version="1.0" encoding="UTF-8" ?>
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
MajorVersion="1"
MinorVersion="0" AssertionID="random-182726" Issuer="sunserver.example.com"
IssueInstant="2001-11-05T17:23:00GMT-02:00">
        <saml:AuthenticationStatement
AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
AuthenticationInstant="2001-11-05T17:22:00GMT-02:00">
            <saml:Subject>
                <saml:NameIdentifier NameQualifier="example.com">John
Doe</saml:NameIdentifier>
            </saml:Subject>
        </saml:AuthenticationStatement>
</saml:Assertion>
```

- An *attribute assertion* declares that the specified subject is associated with the specified attribute. In Access Manager, the Identity Management module is the attribute authority.
- An *authorization decision assertion* declares that the specified subject's request for access to a specified resource has been granted or denied. In Access Manager, the Policy Service is the authorization authority.

One assertion may contain many different statements made by the authority.

SAML SOAP Receiver

Assertions are exchanged between Access Manager and inquiring parties using the `request` and `response` XML-based protocol defined in the SAML specification. These SAML assertions are then integrated into a standard communication protocol for transport purposes.

NOTE Access Manager uses SOAP, a message communications specification integrating XML and HTTPS, to transport requests and responses in its "Web Browser Artifact Profile" on page 282.

SOAP binding defines how SAML `request` and `response` message exchanges are integrated into SOAP exchanges. The SAML SOAP Receiver is a servlet that processes the message. It receives a SOAP message, extracts the SAML request and responds with another SOAP message containing the requested assertion. It responds to queries for authentication, attributes or authorization decisions as well as those that include an assertion identifier reference or artifact by returning assertions.

NOTE The access URL for the SAML SOAP Receiver is `http(s)://`*identity_server_host.domain_name*`:`*port*`/`*server_deploy_uri*`/SAMLSOAPReceiver`. The SAML SOAP Receiver only supports the POST method.

SOAP Messages

SOAP messages consist of three parts: an envelope, header data and a message body. (The SAML `request`/`response` elements are enclosed in the message body.) A client, acting as a SAML requestor, transmits a `<Request>` element within the body of a SOAP message to an entity acting as a SAML Receiver. In answer, the SAML Receiver MUST return either a `<Response>` element within the body of another SOAP message or a SOAP fault code (or error message).

NOTE The SAML requestor and the SAML Receiver MUST NOT include more than one SAML request or response per SOAP message or any additional XML elements in the SOAP body.

A SAML Request may contain queries for any of the following: authentication status, authorization decisions, attribute information and one or more assertion identifier references or artifacts. A SAML Response is sent back to the requesting party for every Request received.

NOTE The SAML SDK and the Java API for XML Messaging (JAXM) are used to construct SOAP messages and send them to the SOAP Receiver.

Protecting The SOAP Receiver

The Access Manager administrator has the option of protecting the SAML SOAP Receiver using authentication. The available methods are:

- NOAUTH
- BASICAUTH

- SSL
- SSLWITHBASICAUTH

This option is configured in the Trusted Partner Sites attribute of the SAML Service in the form:

`SourceID=`*source_id_of_site*`|SOAPUrl=`*url_of_site*`|AuthType=`*chosen_auth_option*`|User=`*user_id*

NOTE The value `user=`*user_id* is used only with the Basic Authentication and SSL With Basic Authentication options.

The default authentication type is NOAUTH. If SSL authentication is to be specified, it is configured in the `SOAPUrl` field with the `https` URL prefix. More information on the Trusted Partner Sites and other SAML Service attributes can be found in the SAML Attributes chapter of the *Sun Java System Access Manager Administration Guide*.

amSAML.xml

`amSAML.xml` is the XML service file that defines the attributes for the SAML Service. All of the attributes in the SAML Service can be managed through either the Access Manager console or the XML service file except two. These attributes can only be managed through `amSAML.xml` using the `amadmin` command line interface.

- `iplanet-am-saml-cleanup-interval` is used to specify how often the internal thread is run in order to cleanup expired assertions from the internal data store. The default is 180 seconds.
- `iplanet-am-saml-assertion-max-number` is used to specify the maximum number of assertions the server can hold at one time. No new assertion will be created if the maximum number is reached. The default value is 0 which means there is no limit.

To change the values of these attributes, the `amSAML.xml` service file needs to be modified, the old `amSAML.xml` service file needs to be deleted, and the newly modified file reloaded using `amadmin`. Information on how to use `amadmin` can be found in The amadmin Command Line Tool chapter of the *Sun Java System Access Manager Administration Guide*. Information on the other SAML Service attributes can also be found in the *Sun Java System Access Manager Administration Guide*.

SAML SDK

Access Manager contains a SAML SDK made up of several Java packages. Administrators can use these packages to integrate the SAML functionality and XML messages into their applications and services. The SDK supports all types of assertions and operates with the Access Manager authorities to process external SAML requests and generate SAML responses. The packages include:

- `com.sun.identity.saml`
- `com.sun.identity.saml.assertion`
- `com.sun.identity.saml.common`
- `com.sun.identity.saml.plugins`
- `com.sun.identity.saml.protocol`
- `com.sun.identity.saml.xmlsig`

com.sun.identity.saml

This package contains the `AssertionManager` and `SAMLClient` classes. The `AssertionManager` provides interfaces and methods to create and get assertions, authentication assertions and assertion artifacts; it is the connection between the SAML specification and the Access Manager. Some of the methods included are:

- `createAssertion`—creates an assertion with an authentication statement based on an Access Manager SSO Token ID.
- `createAssertionArtifact`—creates an artifact that references an assertion based on an Access Manager SSO Token ID.
- `getAssertion`—returns an assertion based on the given parameter (given artifact, assertion ID or query).

The `SAMLClient` provides methods to execute either the Artifact or POST profile from within an application as opposed to a web browser. Its methods include:

- `getAssertionByArtifact`—returns an assertion for a corresponding artifact.
- `doWebPOST`—is designed to do the SAML web-browser POST profile.
- `doWebArtifact`—is designed to do the SAML web-browser profile with artifact.

com.sun.identity.saml.assertion

This package contains the classes needed to create, manage, and integrate, an XML assertion into an application. For example, Code Example 11-2 illustrates how to use the Attribute class and `getAttributeValue` method to get the value of an attribute. From an Assertion, call the `getStatement()` method to retrieve a set of statements. If a statement is an `AttributeStatement`, call the `getAttribute()` method to get a list of attributes. From there, call `getAttributeValue()` to retrieve the AttributeValue.

Code Example 11-2 Sample Code To Get An Attribute Value

```
// get statement in the assertion
Set set = assertion.getStatement();
//assume there is one AttributeStatement
//should check null& instanceof
AttributeStatement statement = (AttributeStatement) set.iterator().next();
List attributes = statement.getAttribute();
// assume there is at least one Attribute
Attribute attribute = (Attribute) attributes.get(0);
List values = attribute.getAttributeValue();
```

com.sun.identity.saml.common

This package defines classes common to all SAML elements including site_ID, issuer name and server host. It also contains all SAML-related exceptions.

CAUTION The date format, `yyyy-MM-dd'T'HH:mm:ss'+/-'HH:mm` , which was used in JDK 1.3.1 with IS 6.0 is no longer supported in IS 6.1. The correct format in JDK 1.4.1 for use in Access Manager 6.1 is:

`yyyy-MM-dd'T'HH:mm:ss'+/-'HHmm`

or

`yyyy-MM-dd'T'HH:mm:ss'GMT''+/-'HH:mm`

For example, the following are correct:

2003-04-22T01:20:02 -0001 (with a space before the zone sign)

2003-04-22T01:20:02GMT-00:01

2003-04-22T01:20:02-0001

com.sun.identity.saml.plugins

Access Manager provides four SPIs, three of them with default implementations. The implementations of these SPIs can be altered, or brand new ones written, based on the specifications of a particular customized service. These can then be used to integrate the SAML Service into the custom service. Currently, the APIs include the `AccountMapper`, `ActionMapper`, `AttributeMapper` and `SiteAttributeMapper`.

- `AccountMapper` is used to map external partner site user accounts to Access Manager user accounts for purposes of single sign-on. A default account mapper implementation is provided. If a site-specific account mapper is not configured, this default mapper is used.

NOTE The default account mapper class is `com.sun.identity.saml.plugin.DefaultAccountMapper`.

  For example, assume the single sign-on is configured from site A to site B, then a site-specific account mapper can be developed and added to site B's Trusted Partner Sites listing in this format:

  `sourceid=`*site_A_source_id* `| accountmapper=`*class_name_of_site specific_account_mapper* `| ...`

  When site B processes the assertion received through either SAML profile, it finds out the source ID of the originating site and locates the account mapper corresponding to that site.

NOTE Turning on the Debug Service in `AMConfig.properties` file, would log additional information concerning the account mapper. For example, was it loaded or what is the user name and organization to which it has been mapped. Information on this can be found in Appendix A, "AMConfig.properties File," in this manual.

- `AttributeMapper` is used in the `AttributeQuery` case. When a site receives an `AttributeQuery`, this mapper is called to obtain the `SSOToken` or an Assertion containing `AuthenticationStatement` from the query. It is also used to convert the attribute in the query to an attribute Access Manager understands. A default attribute mapper is provided. A site-specific attribute mapper can be developed in this format:

  `sourceid=`*site_source_id* `|`
  `attributemapper=`*class_name_of_site_specific_attribute_mapper* `|...`

- `ActionMapper` is used to get SSO information and to map partner actions to Access Manager authorization decisions. A default action mapper implementation is provided. If a site-specific action mapper is not supplied, this default mapper is used. A site-specific action mapper can be developed in this format:

  `sourceid=`*site_source_id* `|`
  `actionmapper=`*class_name_of_site_specific_action_mapper*`|...`

- `SiteAttributeMapper` is also used for SSO. The default functionality of Access Manager is that when no mapper is specified and an assertion is created, either through the web browser Artifact or POST profiles, it only contains `AuthenticationStatement(s)`. If a site wants to include `AttributeStatement(s)`, it can use this SPI to obtain the attributes. It creates `AttributeStatement(s)` from those attributes, and puts them inside the assertion. A site attribute mapper can be developed in this format:

  `sourceid=`*site's source ID* `|`
  `siteattributemapper=`*class_name_of_site_specific_siteattribute_mapper*`|...`

NOTE The default behavior is that no attribute statements are returned unless specified in the plug-in.

com.sun.identity.saml.protocol

This package contains classes that parse the request and response XML messages used to exchange assertions and their authentication, attribute or authorization information.

AuthenticationQuery

The `AuthenticationQuery` class represents an authentication query. An application sends a SAML request with an `AuthenticationQuery` inside. The Subject of the `AuthenticationQuery` must contain a SubjectConfirmation element. In this element, ConfirmationMethod needs to be set to `urn:com:sun:identity`, and SubjectConfirmationData needs to be set to the SSOToken id of the Subject. If the Subject contains a NameIdentifier, then the info in the NameIdentifier should be the same as the one in the SSOToken.

AttributeQuery

The `AttributeQuery` class represents a query concerning an identity's attributes. An application sends a SAML request with an `AttributeQuery` inside. The application develops an `AttributeMapper` to obtain either a SSOToken ID or an Assertion containing an AuthenticationStatement from the query and the mapper is then used to retrieve the attributes for the Subject. If no `AttributeMapper` for the querying site is found, then the `DefaultAttributeMapper` will be used. To use the `DefaultAttributeMapper`, the application should put either the SSOToken ID or an assertion containing an AuthenticationStatement in the `SubjectConfirmationData` element of the Subject in the query. If an SSOToken ID is used, then the ConfirmationMethod must be set to `urn:com:sun:identity:`. If an assertion is used, then this assertion should be issued by the Access Manager instance processing the query or a server that is trusted by the Access Manager instance processing the query.

NOTE In `DefaultAttributeMapper`, it is possible to query a subject's attributes using another subject's SSOToken as long as the SSOToken has the privilege of retrieving those attributes.

For a query using the `DefaultAttributeMapper`, any matching attributes found in the Identity Management module will be returned. If no AttributeDesignator is specified in the AttributeQuery, all attributes from the services defined under the `userServiceNameList` in `amSAML.properties` will be returned. `userServiceNameList`'s value is user service names separated by a comma.

AuthorizationDecisionQuery

The `AuthorizationDecisionQuery` class represents a query concerning an identity's authority to access protected resources. An application sends a SAML request with an `AuthorizationDecisionQuery` inside. The application develops an `ActionMapper` to obtain an SSOToken ID. The mapper is then used to retrieve the authentication decisions for the actions defined in the query.

If no `ActionMapper` for the querying site is found in the configuration, a `DefaultActionMapper` will be used. To use the `DefaultActionMapper`, the application should put the SSOToken ID in the SubjectConfirmationData element of the Subject in the query. If SSOToken ID is used, then the ConfirmationMethod must be set to `urn:com:sun:identity:`. If a NameIdentifier is present, then the info in the SSOToken must be the same as the one in the NameIdentifier.

NOTE The `DefaultActionMapper` handles actions in action namespace `urn:oasis:names:tc:SAML:1.0:ghpp` only. The `iPlanetAMWebAgentService` is used to serve the policy decisions for this action namespace.

The application may also pass in the authentication information through the Evidence element in the query. The Evidence could be an AssertionIDReference or an assertion containing an AuthenticationStatement issued by the Access Manager instance processing the query, or an assertion issued by a server that is trusted by the Access Manager instance processing the query. The Subject in the AuthenticationStatement as the evidence should be the same as the one in the query.

NOTE Policy conditions can be passed in through AttributeStatements of Assertion(s) inside the Evidence of the query. If the value of an attribute contains TEXT node only, then the condition is set as `attributeName=attributeValueString`; otherwise, the condition is set as `attributename=attributeValueElement`.

AuthorizationDecisionQuery Sample

There are many ways to form an authorization decision query and have the decision assertion returned. Code Example 11-3 illustrates one way to do it.

Code Example 11-3 AuthorizationDecisionQuery Code Sample

```
        // testing getAssertion(authZQuery): no SC, with ni, with
        // evidence(AssertionIDRef, authN, for this ni):
            String nameQualifier = "dc=iplanet,dc=com";
            String pName = "uid=amadmin,ou=people,dc=iplanet,dc=com";
            NameIdentifier ni = new NameIdentifier(pName, nameQualifier);
            Subject subject = new Subject(ni);
            String actionNamespace = "urn:test";
            // policy should be added to this resource with these
            // actions for the subject
            Action action1 = new Action(actionNamespace, "GET");
            Action action2 = new Action(actionNamespace, "POST");
            List actions = new ArrayList();
            actions.add(action1);
            actions.add(action2);
            String resource = "http://www.sun.com:80";
            eviSet = new HashSet();
            // this assertion should contain authentication assertion for
            // this subject and should be created by a trusted server
            eviSet.add(eviAssertionIDRef3);
            evidence = new Evidence(eviSet);
```

Code Example 11-3 AuthorizationDecisionQuery Code Sample *(Continued)*

```
            authzQuery = new AuthorizationDecisionQuery(eviSubject1, actions,
                                     evidence, resource);
             try {
                 assertion = am.getAssertion(authzQuery, destID);
             } catch (SAMLException e) {
                 out.println("--failed. Exception:" + e);
             }
```

com.sun.identity.saml.xmlsig

All SAML assertions, requests and responses may be signed using this signature API. This is an SPI in which the interfaces can be implemented and proprietary XML / signature implementations can be plugged in. This package contains the classes needed to sign and verify. By default, the keystore provided with the JDK is used and the key type is DSA. The configuration properties for this functionality are in AMConfig.properties. Information on these properties can be found in "SAML" on page 342 of Appendix A, "AMConfig.properties File." See "SAML Samples" for information on the signature functionality.

SAML Samples

There are several samples that can be accessed from the Access Manager installation. They are located in *IdentityServer_base*/`SUNWam/samples/saml`. These samples illustrate how the SAML service can be used in different ways. They include:

- A sample that serves as the basis for using the SAML client API. This sample is located in *IdentityServer_base*/`SUNWam/samples/saml/client`.
- A sample that illustrates how to form a Query, and write an `AttributeMapper` as well as how to send and process a SOAP message using the SAML SDK. This sample is located in *IdentityServer_base*/`SUNWam/samples/saml/query`.
- A sample application for achieving SSO using either the Web Browser Artifact or the Web Browser POST profiles. This sample is located in *IdentityServer_base*/`SUNWam/samples/saml/sso`.
- A sample that illustrates how to use the XMLSIG API. It details how to configure for XML signing and is located in *IdentityServer_base*/`SUNWam/samples/saml/xmlsig`.

A README file is included with each sample with information and instructions on how to use it.

Chapter 12

Auditing Features

Sun Java™ System Access Manager 6 2005Q1 provides a Logging Service to record information such as user activity, traffic patterns, and authorization violations. The Logging API allow external applications to take advantage of the Logging Service. In addition, the debug files allow administrators to troubleshoot their installation. This chapter explains these auditing features. It contains the following sections:

Logging Service Overview

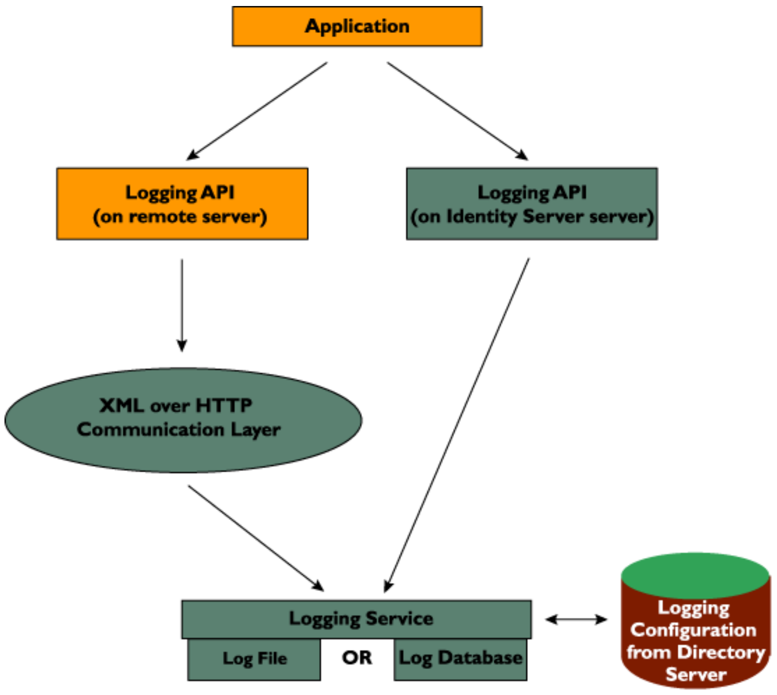
The Logging Service enables all Access Manager services to record information that might be useful to the administrator in one centralized location. The information may include access denials and approvals, authorization violations and code exceptions. Logging allows administrators to analyze user activity, Access Manager traffic patterns and authorization violations. As with all Access Manager services, the Logging Service uses a global service configuration file, named `amLogging.xml`, to define its attributes (such as maximum log size and log location, or whether the log information is written to a flat file or a relational database). The default location for all log files is `/var/opt/SUNWam/logs`.

NOTE This default log directory can be reconfigured after installation by modifying the Log Location attribute in the Logging Service. More information can be found in the Logging Service Attributes chapter in the *Sun Java System Access Manager Administration Guide.*

Logging Architecture

Java applications use the Logging API to access the Logging Service. These interfaces may reside on a remote server or on the same server as Access Manager. An application accesses the Logging Service by calling the Logging API. (If remote, the API uses a XML over HTTP layer to send the logging request to the Logging Service.) The Access Manager SDK loads the configuration data (stored in Directory Server) into the Logging Service when Access Manager starts up or when any logging configuration data is changed via the console. This data includes the log message format, log file name, maximum log size, and the number of history files. Any exception message will be logged, based on the configuration values. Figure 12-1 illustrates the architecture of the Logging Service.

Figure 12-1 Logging Service Architecture


Application
Logging API (on remote server)
Logging API (on Identity Server server)
XML over HTTP Communication Layer
Logging Service
Log File
OR
Log Database
Logging Configuration from Directory Server

amLogging.xml

The Logging Service holds the attributes and values for the logging function. These attributes and values are defined in the `amLogging.xml` service file located in `/etc/opt/SUNWam/config/xml`. These values are applied across the Access Manager deployment and inherited by every configured organization. The structure of `amLogging.xml` is defined by the `sms.dtd`. Information on this document can be found in "The sms.dtd Structure" on page 199 of Chapter 8, "Service Management." Specific information on the Logging Service attributes can be found in the Logging Service Attributes chapter in the *Sun Java System Access Manager Administration Guide.*

Log Files

The log files record a number of events for each of the services it monitors. These files should be checked by the administrator on a regular basis. The default directory for the log files is `/var/opt/SUNWam/logs`. The log file directory can be configured in the Logging Service by using the Access Manager console.

Table 12-1 describes the files in the logs directory. Note that in database formats, the period (.) separator is converted to an underscore (_). 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`.

Table 12-1 Log Files

File or Table	Information Logged
`amAuthLog`	Policy denies
`amPolicy.access`	Policy allows
`amConsole.access`	console events
`amAuthentication.access`	authentication successes
`amAuthentication.error`	authentication failures
`amPasswordReset.access`	password reset events
`amSSO.access`	SSO creates/destroys
`amSAML.access`	SAML successful events
`amSAML.error`	SAML error events
`amLiberty.access`	Liberty successful events

Table 12-1 Log Files

File or Table	Information Logged
`amLiberty.error`	Liberty error events
`amFederation.access`	Federation successful events
`amFederation.error`	Federation error events
`amAdmin.access`	amadmin CLI successful events
`amAdmin.error`	amadmin CLI error events

Recorded Events

The Logging Service logs information passed to the `LogRecord` class by the client. Out-of-the-box, the contents of the `LogRecord` that will be logged are:

Time

This record is the date (`YYYY-MM-DD`) and time (`HH:MM:SS`) at which the log message was recorded.

Data

This record details the description of the user activity, errors or other useful information which the application wants to log.

ModuleName

This record is the 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 314.

Domain

This field records the Access Manager domain to which the user belongs.

Log Level

This record corresponds to the Java 2 Platform, Standard Edition (J2SE) version 1.4 log level of the log record.

Login ID

This field is the ID of the user attempting to access the application. The information (the user to whom the log information belongs) is taken from the session token.

IP Address

This field records the IP address from which the operation was performed.

Logged By

This field is the user who writes the log record. The information is taken from the session token passed during `logger.log(logRecord, ssoToken)`.

Host Name

This field is the host name from which the operation was performed.

Additional fields can also be logged. The new field names must first be added to the `amLogging.xml` service file and the modified service file then reloaded into the Directory Server. The new values for these fields would then be included in the LogRecord Class passed to the Logging Service. More information on how to modify and load an XML service file can be found in "Defining A Custom Service" on page 187 of Chapter 8, "Service Management."

NOTE Only the flat file format can accommodate new logging fields. Other formats might contain steps not documented here. An example would be the database table where a new column must also be added to the table.

Log File Formats

Access Manager can record events in flat text files or a relational database. (The JDK SPI allows extending existing handlers or adding new ones.)

Flat File Format

The default flat file format is the W3C Extended Log Format (ELF). In leveraging this format, the Logging Service records the default logging fields in each log record. Code Example 12-1 illustrates an authentication log record formatted for a flat file. In order, the fields for these values are TIME, DATA, MODULENAME, DOMAIN, LOGLEVEL, LOGINID, IPADDR, LOGGEDBY, and HOSTNAME.

Code Example 12-1 Flat File Record From amAuthentication.access

```
"08-07-2003 07:58:26"   "Login Success service->adminconsoleservice"    LDAP
dc=example,dc=com       INFO    uid=amAdmin,ou=People,dc=example,dc=com
129.149.247.58 "cn=dsameuser,ou=DSAME Users,dc=example,dc=com"
cache1nwk.SFBay.Sun.COM
```

Relational Database Format

For Java applications using a relational database to log messages, the message is stored in a database table. Access Manager uses Java Database Connectivity (JDBC) to access the data. Oracle® and MySQL databases are currently supported.

NOTE JDBC technology is an API for accessing tabular data source using Java. It provides connectivity to a wide range of SQL databases, and access to other tabular data sources, such as spreadsheets or flat files.

Table 12-2 contains the schema for a relational database.

Table 12-2 Relational Database Log Format

Column Name	Data Type	Description
`TIME`	VARCHAR2(30)	Date of the log in the format `YYYY-MM-DD HH:MM:SS`.
`DATA`	VARCHAR2(1024)	The log message itself.
`MODULENAME`	VARCHAR2(255)	The name of the Access Manager service invoking the log record.
`DOMAIN`	VARCHAR2(255)	Access Manager domain of the user.
`LOGLEVEL`	VARCHAR2(255)	JDK 1.4 log level of the log record.
`LOGINID`	VARCHAR2(255)	Login ID of the user who performed the logged operation.
`IPADDR`	VARCHAR2(255)	IP Address of the machine from which the logged operation was performed.
`LOGGEDBY`	VARCHAR2(255)	Login ID of the user who writes the log record.
`HOSTNAME`	VARCHAR2(255)	Host name of machine from which the logged operation was performed.

Oracle Database

In order to log to an Oracle database, the Log Location attribute in the Access Manager Logging Service and the driver variable in the database itself need to be modified. Using the Access Manager console, change the value of the Log Location attribute to:

`jdbc:oracle:thin:@`*hostname*`:1521:`*database_name*

In the database itself, change the value for the driver to:

```
oracle.jdbc.driver.OracleDriver
```

MySQL Database

In order to log to an MySQL database, the Log Location attribute in the Access Manager Logging Service and the driver variable in the database itself need to be modified.

NOTE There is a limitation in the data length for MySQL JDBC logging as MySQL does not support data of more than 255 characters.

Using the Access Manager console, change the value of the Log Location attribute to:

`jdbc:mysql://`*hostname*:*port*/*database_name*

In the database itself, change the value for the driver to:

```
com.mysql.jdbc.Driver
```

CAUTION When MySQL is installed on Solaris or other Unix platforms and modifications are made to the Logging Service, logging into the MySQL database shows the warning message *Syntax error or access violation*.

Java Enterprise System Installation Logs

Events recorded during installation are stored in `/var/sadm/install/logs`. As Access Manager is installed via Java Enterprise System (JES), the events are recorded by the JES installer. The four installation logs are:

- Java_Enterprise_System_Config_Log
- Java_Enterprise_System_Summary_Report_install
- Java_Enterprise_System_install

- Java_Enterprise_System_shared_component_install

Access Manager Service Logs

There are two different types of service log files: access and error. Access log files record general auditing information concerning the deployment (successful or failed authentications, new federations, etc.). Error log files record errors that occur within the application. 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 would be named `amConsole.access` while a database column logging the same events would be called `AMCONSOLE_ACCESS`. The following sections describe the log files recorded by the Logging Service.

Session Logs

The Logging Service records the following events for the Session Service:

- `Login`
- `Logout`
- `Session Idle TimeOut`
- `Session Max TimeOut`
- `Failed To Login`
- `Session Reactivation`
- `Session Destroy`

The session logs are prefixed with `amSSO`.

Console Logs

The Access Manager console logs record the creation, deletion and modification of identity-related objects, policies and services including, among others, organizations, organizational units, users, roles, policies and groups. It also records modifications of user attributes including passwords and the addition or removal of users to or from roles and groups. The console logs are prefixed with `amConsole`.

Authentication Logs

The Authentication component logs user logins and logouts. The authentication logs are prefixed with `amAuthentication`.

Federation Logs

The Federation component logs federation-related events including, but not limited to, the creation of an Authentication Domain and the creation of a Hosted Provider. The federation logs are prefixed with `Federation`.

Policy Logs

The Policy component records policy-related events including, but not limited to, policy administration (policy creation, deletion and modification) and policy evaluation. The policy logs are prefixed with `amPolicy`. Code Example 12-2 on page 305 is a collection of sample records that might appear in the policy logs.

Code Example 12-2 Sample Policy Log Records

```
#Fields: time   Data    ModuleName      Domain  LogLevel        LoginID
IPAddr LoggedBy        HostName

"08-07-2003 11:08:19"   "Created policy test successfully in
Organization dc=iplanet,dc=com"    amPolicy.access "Not Available"
INFO    uid=amAdmin,ou=People,dc=iplanet,dc=com /192.18.120.236
uid=amAdmin,ou=People,dc=iplanet,dc=com 192.18.120.236

"08-07-2003 11:08:55"   "Modified policy test successfully in
Organization dc=iplanet,dc=com"   amPolicy.access "Not Available"
INFO    uid=amAdmin,ou=People,dc=iplanet,dc=com /192.18.120.236
uid=amAdmin,ou=People,dc=iplanet,dc=com 192.18.120.236

"08-07-2003 11:09:05"   "Removed policy test successfully in
Organization dc=iplanet,dc=com"    amPolicy.access "Not Available"
INFO    uid=amAdmin,ou=People,dc=iplanet,dc=com /192.18.120.236
uid=amAdmin,ou=People,dc=iplanet,dc=com 192.18.120.236

"08-07-2003 11:15:43"   "Policy Evaluation result of Policy test in
Organization dc=iplanet,dc=com for service iPlanetAMWebAgentService,
resource http://moonshadow.red.iplanet.com:80/*.html and action names
[GET, POST] is GET=[allow]\\n." amPolicy.access "Not Available" INFO
uid=amAdmin,ou=People,dc=iplanet,dc=com /192.18.120.236
uid=amAdmin,ou=People,dc=iplanet,dc=com 192.18.120.236
```

Agent Logs

The policy agent logs are responsible for logging exceptions regarding log resources that were either allowed or denied to a user. The agent logs are prefixed with `amAgent`. `amAgent` logs reside on the agent server only. Agent events are logged on the Access Manager machine in the Authentication Logs. For more information on this function, see the correct documentation for the policy agent in question.

SAML Logs

The SAML component records SAML-related events including, but not limited to, assertion and artifact creation or removal, response and request details, and SOAP errors. The session logs are prefixed with `amSAML`.

amAdmin Logs

The command line logs record event errors that occur during operations using the command line tools. These include, but are not limited to, loading a service schema, creating policy and deleting users. The command line logs are prefixed with `amAdmin`. More information can be found in "Command Line Logging" on page 307.

Logging Features

The Logging Service has a number of special features which can be enabled for additional functionality. They include To Enable Secure Logging, Command Line Logging and Remote Logging.

To Enable Secure Logging

This optional feature adds additional security to the logging function. Secure Logging enables detection of unauthorized changes to, or tampering of, the security logs. No special coding is required to leverage this feature. Secure Logging is accomplished by using a pre-registered certificate configured by the system administrator. This Manifest Analysis and Certification (MAC) is generated and stored for every log record. A special "signature" log record is periodically inserted that represents the signature for the contents of the log written to that point. The combination of the two records ensures that the logs have not been tampered with. Secure Logging can be enabled by performing the following steps:

1. Create a certificate with the name *Logger* and install it in the deployment container running Access Manager.

   Refer to the documentation that comes with the deployment container for details.

2. Turn on Secure Logging in the Logging Service configuration using the Access Manager console and save the change.

   The administrator can also modify the default values for the other attributes in the Logging Service.

3. Create a file in the *IdentityServer_base*/SUNWam/config directory that contains the certificate database password and name it .wtpass.

NOTE The file name and the path to it is configurable in the AMConfig.properties file. For more information see the "Certificate Database" on page 339 of Appendix A, "AMConfig.properties File."

Ensure that the deployment container user is the only administrator with read permissions to this file for security reasons.

4. Restart the server after making these changes.

Command Line Logging

The amadmin command line tool has the ability to create, modify and delete identity objects (organizations, users, and roles, for example) in Directory Server. This tool can also load, create, and register service templates. The Logging Service can record these command line actions by invoking the -t option. If the com.iplanet.am.logstatus property in AMConfig.properties is enabled (ACTIVE) then a log record will be created. (This property is enabled by default.) The command line logs are prefixed with amAdmin. More information can be found in Chapter 8, "The amadmin Command Line Tool" in the *Sun Java System Access Manager Administration Guide*.

Remote Logging

Access Manager supports remote logging. This allows a client using the Access Manager SDK to create log records on an instance of Access Manager deployed on a remote machine.

Using Remote Logging

Remote logging can be initiated in any of the following scenarios:

- When the logging URL in the Naming Service of one Access Manager instance points to a remote instance and there is a trust relationship configured between the two, logs will be written to the remote Access Manager instance.

- When the Access Manager SDK is installed against a remote Access Manager instance and a client (or a simple Java class) running on the SDK server uses the logging APIs, the logs will be written to the remote Access Manager machine.
- When logging APIs are used by Access Manager agents.

Enabling Remote Logging

To enable remote logging, ensure that the following information is regarded.

- If using Sun Java System Web Server, the following environment variables need to be set in the `server.xml` configuration file.

  a. `java.util.logging.manager=com.sun.identity.log.LogManager`

  b. `java.util.logging.config.file=/`*IdentityServer_base*`/SUNwam/lib/LogConfig.properties`

  - If the Java™ 2 Platform, Standard Edition being used is 1.4 or later, this is accomplished by invoking the following at the command line:

    ```
    java -cp
    /IdentityServer_base/SUNWam/lib/am_logging.jar:/IdentityServer_base/S
    UNWam/lib/xercesImpl.jar:/IdentityServer_base/SUNWam/lib/xmlPars
    erAPIs.jar:/IdentityServer_base/SUNWam/lib/jaas.jar:/IdentityServer_
    base/SUNWam/lib/xmlParserAPIs.jar:/IdentityServer_base/SUNWam/lib
    /servlet.jar:/IdentityServer_base/SUNWam/locale:/IdentityServer_base/
    SUNWam/lib/am_services.jar:/IdentityServer_base/SUNWam/lib/am_sd
    k.jar:/IdentityServer_base/SUNWam/lib/jss311.jar:/IdentityServer_base
    /SUNWam/lib:.

    -Djava.util.logging.manager=com.sun.identity.log.LogManager

    -Djava.util.logging.config.file=/IdentityServer_base/SUNwam/lib/
    LogConfig.properties <logTestClass>
    ```

  - If the Java 2 Platform, Standard Edition being used is earlier than 1.4, this is accomplished by invoking the following at the command line:

    ```
    java
    -Xbootclasspath/a:/IdentityServer_base/SUNWam/lib/jdk_logging.ja
    r -cp
    /IdentityServer_base/SUNWam/lib/am_logging.jar:/IdentityServer_base/S
    UNWam/lib/xercesImpl.jar:/IdentityServer_base/SUNWam/lib/xmlPars
    erAPIs.jar:/IdentityServer_base/SUNWam/lib/jaas.jar:/IdentityServer_
    base/SUNWam/lib/xmlParserAPIs.jar:/IdentityServer_base/SUNWam/lib
    ```

`/servlet.jar:/`*IdentityServer_base*`/SUNWam/locale:/`*IdentityServer_base*`/SUNWam/lib/am_services.jar:/`*IdentityServer_base*`/SUNWam/lib/am_sdk.jar:/`*IdentityServer_base*`/SUNWam/lib/jss311.jar:/`*IdentityServer_base*`/SUNWam/lib:.`

`-Djava.util.logging.manager=com.sun.identity.log.LogManager`

`-Djava.util.logging.config.file=/`*IdentityServer_base*`/SUNwam/lib/LogConfig.properties <logTestClass>`

- Ensure that the following parameters are configured in `LogConfig.properties` located in *IdentityServer_base*`/SUNWam/lib`.

   a. `iplanet-am-logging-remote-handler=com.sun.identity.log.handlers.RemoteHandler`

   b. `iplanet-am-logging-remote-formatter=com.sun.identity.log.handlers.RemoteFormatter`

   c. `iplanet-am-logging-remote-buffer-size=1`

   Remote logging supports buffering on the basis of the number of log records. This value defines the log buffer size by the number of records. Once the buffer is full, all buffered records will be flushed to the server.

   d. `iplanet-am-logging-buffer-time-in-seconds=3600`

   This value defines the time-out period in which to invoke the log buffer-cleaner thread.

   e. `iplanet-am-logging-time-buffering-status=OFF`

   This value defines whether log buffering (and the buffer-cleaner thread) is enabled or not. By default this feature is turned off.

Logging API

The Logging API provides log management tools for all Access Manager services as well as providing a set of Java classes for external applications to create, retrieve, submit, or delete log information. The Access Manager Logging API extend the core logging API in the Java™ 2 Standard Edition Development Kit (JDK) 1.4. Only the `Logger` and `LogRecord` classes are enhanced. They are contained in the package `com.sun.identity.log`.

TIP An overview of the JDK 1.4 logging function can be found at `http://java.sun.com/j2se/1.4.1/docs/guide/util/logging/overview.html` The Javadocs for the JDK 1.4 logging API themselves can be found at `http://java.sun.com/j2se/1.4.1/docs/api/java/util/logging/package-summary.html`.

Setting Environment Variables

The following shared library environment variables must be set in the executable for an application that is using the Logging Service. You can determine how to set the variables depending upon three things:

- Whether the application can execute in the local Access Manager server, or executes only a in remote server
- Whether or not you want the Access Manager LogManager class to override the native LogManager class
- Whether or not SSL is enabled in your deployment

If Client Can Execute in the Local Access Manager Server

When the client application can execute in either the local Access Manager server JVM or in a remote server JVM, choose one of the following two configurations:

- If it is acceptable for the native LogManager class to be overridden by the Access Manager LogManager class in the JDK1.4 environment, then follow these steps:

  a. Set the following variables:

  ```
  -D"java.util.logging.manager=com.sun.identity.log.LogManager"
  ```

  ```
  -D"java.util.logging.config.class=com.sun.identity.log.s1is.
      LogConfigReader"
  ```

  b. If you are using JDK1.3, add the following:

  ```
  -Xbootclasspath/a:/IdentityServer_base/SUNWam/lib/
      jdk_logging.jar
  ```

  In this case, Access Manager will automatically determine whether to configure regular or remote handlers. Note also that Access Manager must be able to access Directory Server in this case.

- If it is *not* acceptable for the native LogManager class to be overridden by the Access Manager LogManager class in the JDK1.4 environment, then follow these steps:

  a. Set the following variables:

  ```
  -DLOG_COMPATMODE=Off
  ```

  ```
  -Ds1is.java.util.logging.config.class=com.sun.identity.log.s1is.
      LogConfigReader
  ```

  b. (Optional) If you are using JDK1.3, add the following:

  ```
  -Xbootclasspath/a:/IdentityServer_base/SUNWam/lib/
      jdk_logging.jar
  ```

  In this case, Access Manager will automatically determine whether to configure regular or remote handlers. Note also that Access Manager must be able to access Directory Server in this case.

If Client Executes Only in a Remote Server

When the client application can execute only in a remote server JVM, choose one of the following two configurations:

- If it is acceptable for the native LogManager class to be overridden by the Access Manager LogManager class in the JDK1.4 environment, then follow these steps:

  a. Set the following variables:

  ```
  -Djava.util.logging.manager=com.sun.identity.log.LogManager
  ```

  ```
  -Djava.util.logging.config.file=/IdentityServer_base/SUNwam/lib/
      LogConfig.properties
  ```

b. In `LogConfig.properties`, or in the `logging.properties` file supplied by JDK, set the following properties:

```
iplanet-am-logging-remote-handler=com.sun.identity.log.handlers.
    RemoteHandler

iplanet-am-logging-remote-formatter=com.sun.identity.log.
    handlers.RemoteFormatter

iplanet-am-logging-remote-buffer-size=1

iplanet-am-logging-buffer-time-in-seconds=3600

iplanet-am-logging-time-buffering-status=OFF
```

c. If you are using JDK1.3, add the following:

```
-Xbootclasspath/a:/IdentityServer_base/SUNWam/lib/
    jdk_logging.jar
```

In this case, the Logging API will configure a remote handler for all logs. Note that access to Directory Server is not required in this case.

- If it is *not* acceptable for the native LogManager class to be overridden by the Access Manager LogManager class in the JDK1.4 environment, then follow these steps:

a. Set the following variables:

```
-DLOG_COMPATMODE=Off

-Ds1is.java.util.logging.config.file=/IdentityServer_base/
    SUNwam/lib/LogConfig.properties
```

b. In `LogConfig.properties`, or in the `logging.properties` file supplied by JDK, set the following properties:

```
iplanet-am-logging-remote-handler=com.sun.identity.log.handlers.
    RemoteHandler

iplanet-am-logging-remote-formatter=com.sun.identity.log.
    handlers.RemoteFormatter

iplanet-am-logging-remote-buffer-size=1

iplanet-am-logging-buffer-time-in-seconds=3600

iplanet-am-logging-time-buffering-status=OFF
```

c. If you are using JDK1.3, add the following:

```
-Xbootclasspath/a:/IdentityServer_base/SUNWam/lib/
    jdk_logging.jar
```

The Client SDK APIs use this logging configuration by default. In this case, the Logging API will configure a remote handler for all logs. Note that access to the Directory Server is not required in this case.

If SSL is Enabled

If SSL is enable and uses JSS for Access Manager, set the following paramter:

```
-D"java.protocol.handler.pkgs=com.iplanet.services.comm"
```

Logger Class

This `Logger` class provides the methods for applications to use in creating log files and writing log information to them.

- The `getLogger()` method returns a logger object and simultaneously creates a log record (`LogRecord`) in the designated logging location.
- The `log()` method records a single piece of log information or a `LogRecord`. It allows an application to submit a logging message to a predetermined log.
    - `Logger.log(logRecord, String credential)` had been added to call the authorization hook. The credential is accepted as a `ssoToken` string. The default authorization hook checks validitity of the `ssoToken`. Data is not logged at all if this check fails.
    - `Logger.log(logRecord)` simply calls `Logger(logRecord, String cred)` with credential value of null. And thus the default authorization check does not allow logging when an application uses this interface.

LogRecord Class

The `LogRecord` class provides the means to represent the information that needs to be logged. Each instance represents a single piece of log information or `logRecord` that comes from the application. The `ssoToken` is passed to the `logRecord` constructor and used to populate the log fields discussed in "Recorded Events" on page 300. The session token passed during the `logger.log(logRecord, ssoToken)` log request is used to authorize the user. The user can only log with a valid `ssoToken`.

Adding Log Data

The following sections illustrate ways to use the Logging API for adding log file information.

Adding ModuleName Data

The ModuleName value can be added to a log file using the `logRecord.addLogInfo(key, value)` API. If a module name is not added, the name of the log will be used to populate this field. For example, authentication information is logged in the `amAuthentication.access` file using an internal session token (`"dsameuser" ssoToken`). If user `Joe123` attempts to authenticate, the LoginID will be Joe123, and the LoggedBy user will be dsameuser.

NOTE The LoggedBy entry is populated from the SSOToken passed during `logger.log(logRecord, ssoToken)` call.

If the authentication module information (such as LDAP, Membership, etc.) is not added by the APIs, `amAuthentication.access` will be the value of the `ModuleName` field.

Adding Log Level Data

A LogLevel is passed in the `LogRecord` constructor using the following code:

```
LogRecord(Level level, String msg)
```

While using the logging APIs, any JDK 1.4 defined log levels can be passed.

Caching Log Records

Access Manager supports log record caching both locally and remotely based on the configurable buffering properties discussed in "Remote Logging" on page 307. Caching is supported for either type of log file although not when secure logging is enabled.

Flushing Log Records

Access Manager provides `Logger.flush()` to expunge all the cached log records.

Sample Logging Code

Code Example 12-3 provides sample code to illustrate one way in which the logging API can be used to write Access Manager records.

Code Example 12-3 Logging API Samples

```
Logger logger = Logger.getLogger("SampleLogFile");
// Creates the file or table in the LogLocation specified in the
amLogging.xml and returns the Logger object.

LogRecord lr = new LogRecord(Level.INFO, "SampleData", ssoToken);
// Creates the LogRecord filling details from ssoToken.

logger.log(lr,ssoToken);
// Writes the info into the backend file, db or remote server.
```

Logging SPI

The Logging SPI are Java packages that can be used to develop plug-ins for customized features. The SPI are organized in the `com.sun.identity.log.spi` package. More information on the SPI can be found in the Javadocs located at *IdentityServer_base*/`SUNWam/docs`.

Log Verifier Plugin

If secure logging is enabled, the log files are verified periodically to detect any attempt of tampering. If tampering is detected, the action taken can be customized by following the steps below.

1. Implement the `com.sun.identity.log.spi.IVerifierOutput` interface with the desired functionality.
2. Add the implementing class in the classpath of Access Manager.
3. Modify the property `iplanet-am-logging-verifier-action-class` in the `/etc/opt/SUNWam/config/xml/amLogging.xml` file with the name of the new class.

Log Authorization Plugin

The Logging Service allows a class to be plugged in that will determine whether a `LogRecord` is logged or discarded based on the authorization of the owner of the session token performing the event.

NOTE The `IAuthorizer` interface accepts a SSOToken and the log record being written.

There are several ways to accomplish this. For example:

1. Get the applicable role or DN of the user from the `SSOToken` and check it against a pre-configured (or hardcoded) list of roles / users that are allowed access. The administrator must configure a role and assign all policy agents and entities (for example, applications) that can possibly log to Access Manager to this role.
2. Instantiate a `PolicyEvaluator` and call `PolicyEvaluator.isAllowed(ssotoken, logname);`. This entails defining a policy XML to model log access and registering it with Access Manager.

In general:

1. Implement the `com.sun.identity.log.spi.IAuthorizer` interface with the desired functionality.
2. Add the implementing class in the classpath of Access Manager.
3. Modify the property `iplanet-am-logging-authz-class` in the `/etc/opt/SUNWam/config/xml/amLogging.xml` file with the name of the new class.

NOTE The Access Manager Javadocs can be accessed from any browser by copying the complete *IdentityServer_base*`/SUNWam/docs/` directory into the *IdentityServer_base*`/SUNWam/public_html` directory and pointing the browser to `http://`*identity_server_host.domain_name*`:`*port*`/docs/index.html`.

Debug Files

The debug files are not a feature of, nor generated by, the Logging Service. They are written using different APIs which are independent of the logging APIs. Debug files are stored in `/var/opt/SUNWam/debug`. This location, along with the level of the debug information, is configurable in the `AMConfig.properties` file, located in the *IdentityServer_base*`/SUNWam/lib/` directory. For more information on the debug properties, see Appendix A, "AMConfig.properties File."

Debug Levels

There are several levels of information that can be recorded to the debug files. The debug level is set using the `com.iplanet.services.debug.level` property in `AMConfig.properties`.

1. Off—No debug information is recorded.
2. Error—This level is used for production. During production, there should be no errors in the debug files.
3. Warning—Currently, using this level is not recommended.
4. Message—This level alerts to possible issues using code tracing. Most Access Manager modules use this level to send debug messages.

CAUTION Warning and Message levels should not be used in production. They cause severe performance degradation and an abundance of debug messages.

Debug Output Files

A debug file does not get created until a module writes to it. Therefore, in the default `error` mode no debug files may be generated. The debug files that get created on a basic login with the debug level set to `message` include:

- amAuth
- amAuthConfig
- amAuthContextLocal
- amAuthLDAP
- amCallback
- amClientDetection
- amConsole
- amFileLookup
- amJSS
- amLog
- amLoginModule
- amLoginViewBean

- amNaming
- amProfile
- amSDK
- amSSOProvider
- amSessionEncodeURL
- amThreadManager

The most often used files are the amSDK, amProfile and all files pertaining to authentication. The information captured includes the date, time and message type (Error, Warning, Message).

Using Debug Files

The debug level, by default, is set to `error`. The debug files might be useful to an administrator when they are:

- Writing a custom authentication module.
- Writing a custom application using the Access Manager SDKs. The `amProfile` and `amSDK` debug files capture this information.
- Troubleshooting access permissions while using the console or SDK. The `amProfile` and `amSDK` debug files also capture this information.
- Troubleshooting SSL.
- Troubleshooting the LDAP authentication module. The `amAuthLDAP` debug file captures this information.

The debug files should go hand in hand with any troubleshooting guide we might have in the future. For example when SSL fails, someone might turn on debug to message and look in the amJSS debug file for any specific cert errors.

Multiple Access Manager Instances And Debug Files

Access Manager contains the `ammultiserverinstall` script that can be used to configure numerous instances of the server. If the multiple server instances are configured to use different debug directories, each individual instance has to have both read and write permissions to the debug directories. More information on the `ammultiserverinstall` script can be found in the *Sun Java System Access Manager Administration Guide*.

Chapter 13

Client Detection Service

The Sun Java™ System Access Manager 6 2005Q1 Authentication Service has the capability of being accessed from many client types, whether HTML-based, WML-based or other protocols. In order for this function to work, Access Manager must be able to identify the client type. The Client Detection Service is used for this purpose. This chapter offers information on the service, and how it can be used to recognize the client type. It contains the following sections:

Overview

The Access Manager Authentication Service has the capability to process requests from multiple browser type clients. Thus, the service can be used to authenticate users attempting to access applications based in HTML, WML or other protocols.

CAUTION The Access Manager console though can not be accessed from any client type except HTML.

The client detection API can be used to determine the protocol of the requesting client browser and retrieve the correctly formatted pages for the particular client type.

NOTE Out of the box, Access Manager only defines client data for supported HTML client browsers. A list of supported browsers can be found in Chapter 1, "Introduction" under the section "Client Browser Support" on page 43.

Client Detection Process

Since any user requesting access to Access Manager must first be successfully authenticated, browser type client detection is accomplished within the Authentication Service. When a client's request is passed to Access Manager, it is directed to the Authentication Service. Within this service, the first step in user validation is to identify the browser type using the `User-Agent` field stored in the HTTP request.

NOTE The `User-Agent` field contains *product tokens* which contains information about the browser type client originating the HTTP request. The tokens are a standard used to allow communicating applications to identify themselves. The format is `software/version library/version`.

The `User-Agent` information is then matched to browser type data defined and stored in the `amClientData.xml` file.

CAUTION User-Agent information is defined in amClientData.xml but this information is stored in Directory Server under Client Detection Service.

Based on this Client Data, correctly formatted browser pages are sent back to the client for authentication (for example, HTML or WML pages). Once the user is validated, the client type is added to the session token (as the key `clientType`) where it can be retrieved and used by other Access Manager services. (If there is no matching client data, the default type is returned.)

NOTE The userAgent must be a part of the client data configured for all browser type clients. It can be a partial string or the exact product token.

Enabling Client Detection

By default, the client detection capability is disabled; this then assumes the client to be of the `genericHTML` type (For example Access Manager will be accessed from a HTML browser). The preferred way to enable the Client Detection Service is to use the Access Manager console and select the option in the Client Detection Service itself. For more information, see the *Sun Java System Access Manager Administration*

Guide. To enable client detection using the `amClientDetection.xml`, the `iplanet-am-client-detection-enabled` attribute must be set to `true`. `amClientDetection.xml` must then be deleted from Directory Server and reloaded using `amAdmin`. The following procedure illustrates the complete enabling process.

1. Import client data XML file using the `amadmin` command
   /IdentityServer_base`/SUNWam/bin/amadmin -u` *amadmin_DN* `-w` *amadmin_password* `-t` *name_of_XML_file*

   This step is only necessary if the client data is not already defined in amClientData.xml. The XML file is based on the "The sms.dtd Structure" on page 199 of Chapter 8, "Service Management."

2. Restart Access Manager.

3. Login to Access Manager console.

4. Go to Service Configuration and click the ClientDetectionproperties.

5. Enable Client Detection.

6. Make sure the imported data can be viewed with Access Manager console.

   Click on the Edit button next to the Client Data attribute.

7. Create a directory for new client type and add customized JSPs.

   Create a new directory in */IdentityServer_base*`/SUNWam/web-src/services/config/auth/default/` and add JSPs for the new client type. Code Example 13-1 on page 323 is a login page written for a WML browser.

Code Example 13-1 Login.jsp Written In WML

```
<?xml version="1.0"?>

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">

<!-- Copyright Sun Microsystems, Inc.  All Rights Reserved -->

<wml>
<head>
<meta http-equiv="Cache-Control" content="max-age=0"/>
</head>

 <card id="authmenu" title="Username">
<do type="accept" label="Enter">

<go method="get" href="/wireless">
<postfield name="TOKEN0" value="$username"/>
```

Code Example 13-1 Login.jsp Written In WML

```
<postfield name="TOKEN1" value="$password"/>
</go>
</do>
<p>
Enter username:
<input type="text" name="password"/>
</p>
<p>
Enter password:
<input type="text" name="username"/>
</p>
</card>
</wml>
```

Client Data

In order to detect client types, Access Manager needs to recognize their identifying characteristics. These characteristics identify the features of all supported types and are defined in the `amClientData.xml` service file. The full scope of client data available is defined as a schema in `amClientData.xml`. The configured Access Manager client data available for HTML-based browsers is defined as sub-configurations of the overall schema: genericHTML and its parent HTML.

NOTE Parent profiles (or *styles*, as they are referred to in the Access Manager console) are defined with properties that are common to its configured child devices. This allows for the dynamic inheritance of the parent properties to the child devices making the device profiles easier to mange.

HTML

`HTML` is a base style containing properties common to HTML-based browsers. It might have several branches including web-based HTML (or genericHTML), cHTML (Compact HTML) and others. All configured devices for this style could inherit these properties which include:

- `parentId`—identifies the base profile. The default value is `HTML`.
- `clientType`—an arbitrary string which uniquely identifies the client. The default value is `HTML`.
- `filePath`—is used to locate the client type files (templates and JSP files). The default value is `html`.

- `contentType`—defines the content type of the HTTP request. The default value is `text/html`.
- `genericHTML`—defines a client that will be treated as HTML. The default value is `true`.

NOTE This attribute does not refer to the similarly named genericHTML style.

- `cookieSupport`—defines whether cookies are supported by the client browser. The default value is `true` which sets a cookie in the response header. The other two values could be `False` which sets the cookie in the URL and `Null` which allows for dynamic cookie detection. In the first request, the cookie is set in both the response header and the URL; the actual mode is then detected and set from the subsequent request.

NOTE Although the Client Detection Service supports a cookieless mode, Access Manager console does not. Therefore, enabling this function will not allow login to the console. This feature is provided for wireless applications and others that will support it.

- `CcppAccept-Charset`—defines the character encoding used by Access Manager to send a response to the browser. The default value is `UTF-8`.

genericHTML

`genericHTML` is a configured device that inherits properties from the HTML style as well as defining its own properties. It refers to a HTML browser (Netscape Navigator™, Microsoft® Internet Explorer, or Mozilla™). Its properties include:

- `parentId`—identifies the base profile for the configured device. The default value is `HTML`.
- `clientType`—an arbitrary string which uniquely identifies the client. The default value is `genericHTML`.
- `userAgent`—a search filter used to compare / match the user agent defined in the HTTP header. The default value is `Mozilla/4.0`.

- `CcppAccept-Charset`—defines the character encoding set supported by the browser. The default values are `UTF-8;ISO-8859-1;ISO-8859-2;ISO-8859-3;ISO-8859-4;ISO-8859-5;ISO-8859-6;ISO-8859-7;ISO-8859-8;ISO-8859-9;ISO-8859-10;ISO-8859-14;ISO-8859-15;Shift_JIS;EUC-JP;ISO-2022-JP;GB18030;GB2312;BIG5;EUC-KR;ISO-2022-KR;TIS-620;KOI8-R`.

NOTE The character set can be configured for any given locale by adding `charset_`*`locale`*`=codeset` where the code set name is based on the Internet Assigned Numbers Authority (IANA) standard.

Client Detection API

Access Manager is packaged with a Java API which can implement the client detection functionality. The client detection API are in a package called `com.iplanet.services.cdm`. This package provides the interfaces and classes needed to retrieve client properties. The client detection procedure would include defining the client type characteristics (as stated in "Client Data" on page 324) as well as implementing the client detection API within the external application.

The client detection capability is provided by `ClientDetectionInterface`, a pluggable interface (not an API invoked by a regular application). It provides a `getClientType` method. The `getClientType` method extracts the client data from the browser's incoming `HttpRequest`, matches the user agent information and returns the `ClientType` as a string. Upon successful authentication, the client type is added to the user's session token. The `ClientDetectionException` handles any error conditions.

Chapter 14

Access Manager Utilities

Sun Java™ System Access Manager 6 2005Q1 provides scripts to backup and restore data as well as application programming interfaces (API) that are used by the server itself or by external applications. This chapter explains the scripts and the API. It contains the following sections:

- "Utility API" on page 327
- "Password API Plug-Ins" on page 329

Utility API

The utilities package is called `com.iplanet.am.util`. It contains utility programs that can be used by external applications accessing Access Manager. Following is a summary of the utility API and their functions.

AdminUtils

This class contains the methods used to retrieve TopLevelAdmin DN and password. The information comes from the server configuration file, `serverconfig.xml`, located in */IdentityServer_base*/`SUNWam/config/ums`.

AMClientDetector

The `AMClientDetector` interface executes the Client Detection Class configured in the Client Detection Service to get the client type.

AMPasswordUtil

The `AMPasswordUtil` interface has two purposes:

1. Encrypting and decrypting any string.
2. Encrypting and decrypting special user passwords such as the password for dsameuser or proxy user.

NOTE Any remote application using this utility should have the value of the AMConfig property `am.encryption.pwd` copied to a properties file on the client side. This value is generated at installation time and stored in /*IdentityServer_base*/`SUNWam/lib/AMConfig.properties`. More information on this property can be found in the Encryption section of the Appendix A, "AMConfig.properties File."

Debug

Debug allows an interface to file debug and exception information in a uniform format. It supports different levels of information (in the ascending order): `OFF`, `ERROR`, `WARNING`, `MESSAGE` and `ON`. A given debug level is enabled if it is set to at least that level. For example, if the debug state is `ERROR`, only errors will be filed. If the debug state is `WARNING`, only errors and warnings will be filed. If the debug state is `MESSAGE`, everything will be filed. `MESSAGE` and `ON` are the same level except `MESSAGE` writes to a file, whereas `ON` writes to `System.out`.

NOTE Debugging is an intensive operation and can hurt performance. Java evaluates the arguments to `message()` and `warning()` even when debugging is turned off. It is recommended that the debug state be checked before invoking any `message()` or `warning()` methods to avoid unnecessary argument evaluation and maximize application performance.

Locale

This class is a utility that provides the functionality for applications and services to internationalize their messages.

SystemProperties

This class provides functionality that allows single-point-of-access to all related system properties. First, the class tries to find `AMConfig.class`, and then a file, `AMConfig.properties`, in the CLASSPATH accessible to this code. The class takes precedence over the flat file. If multiple servers are running, each may have their own configuration file. The naming convention for such scenarios is `AMConfig_serverName`.

ThreadPool

`ThreadPool` is a generic thread pool that manages and recycles threads instead of creating them when a task needs to be run on a different thread. Thread pooling saves the virtual machine the work of creating new threads for every short-lived task. In addition, it minimizes the overhead associated with getting a thread started and cleaning it up after it dies. By creating a pool of threads, a single thread from the pool can be reused any number of times for different tasks. This reduces response time because a thread is already constructed and started and is simply waiting for its next task.

Another characteristic of this thread pool is that it is fixed in size at the time of construction. All the threads are started, and then each goes into a wait state until a task is assigned to it. If all the threads in the pool are currently assigned a task, the pool is empty and new requests (tasks) will have to wait before being scheduled to run. This is a way to put an upper bound on the amount of resources any pool can use up. In the future, this class may be enhanced to provide support growing the size of the pool at runtime to facilitate dynamic tuning.

Password API Plug-Ins

The Password API plug-ins can be used to integrate password functions into applications. They can be used to generate new passwords as well as notify users when their password has been changed. These interfaces are `PasswordGenerator` and `NotifyPassword`, respectively. They can be found in the `com.sun.identity.password.plugins` package.

NOTE The Access Manager Javadocs can be accessed from any browser by copying the complete *IdentityServer_base*/`SUNWam/docs/` directory into the *IdentityServer_base*/`SUNWam/public_html` directory and pointing the browser to `http://`*identity_server_host.domain_name*`:`*port*`/docs/index.html`.

There are samples (which include sample code) for these API that can be accessed from the Access Manager installation. They are located in *IdentityServer_base*/`SUNWam/samples/console`. They include:

Notify Password Sample

This sample details how to build a plug-in which an administrator can define their own method of notification when a user has reset a password. Instructions for this sample are in the `Readme.txt` or `Readme.html` file located in *IdentityServer_base*/`SUNWam/samples/console/NotifyPassword`.

Password Generator Sample

This sample details how to build a plug-in which an administrator can define their own method of random password generation when a user's password is reset using the Password Reset Service. Instructions for this sample are in the `Readme.txt` or `Readme.html` file located in *IdentityServer_base*/`SUNWam/samples/console/PasswordGenerator`.

Appendix A

AMConfig.properties File

`AMConfig.properties` is the resource configuration file for Sun Java™ System Access Manager 6 2005Q1. It provides instructions for the Access Manager deployment. This chapter explains the attributes of `AMConfig.properties`. It contains the following sections:

Overview

Access Manager is configured by placing application properties in plain text configuration files. These configuration files contain one property per line and each has a corresponding value. Properties and their values are case-sensitive. Indentation of the properties is consistent throughout the file. Lines which begin with the characters "/*" are comments, and ignored by the application. Comments are completed with a last line that contains the closing characters "*/". The main configuration file for Access Manager is `AMConfig.properties` located in *IdentityServer_base*/`SUNWam/lib`. The following sections describe the properties and default values of `AMConfig.properties`.

NOTE The Access Manager must be restarted for any modification in `AMConfig.properties` to take effect.

Deployment Properties

Following are the deployment-specific attributes configured in `AMConfig.properties`.

Access Manager

This section describe properties that define the Access Manager application.

Installation

These properties are defined during installation.

- `com.iplanet.am.server.host=`*identity_server_host.domain_name*

  The value of this property is the DNS domain name of the machine on which the Access Manager is located.

- `com.iplanet.am.server.port=58080`

  The value of this property is the port number used by the Access Manager. The default is 58080.

- `com.iplanet.am.jdk.path=/`*IdentityServer_base*`/SUNWam/java`

  The value of this property is the path to the JDK used by the Access Manager.

- `com.sun.identity.authentication.super.user=uid=amAdmin,ou=People,dc=`*top_level_org*`,dc=com`

  This property identifies the full LDAP DN of the super user configured during installation of Access Manager; it is `amadmin` by default. This user must always log in using LDAP authentication as they will always be authenticated against the Directory Server. The UID alone is generally used to login but the full DN as defined in this property can also be used.

Console

These properties are specific to the Access Manager console.

- `com.iplanet.am.console.host=`*identity_server_host.domain_name*

  The value of this property is the DNS domain name of the machine on which the Access Manager console is located.

- `com.iplanet.am.console.protocol=http`

  The value of this property is the protocol used to communicate with the Access Manager. The default is `http`.

- `com.iplanet.am.console.port=58080`

  The value of this property is the port number of the machine on which the Access Manager console is located. The default is 58080.

The following directives can be added to the `AMConfig.properties` file to add their respective functionality to the Access Manager console.

- `com.iplanet.am.console.display.off=orgs,users,groups`

  If specified, Access Manager will not perform the initial search for a specified identity object that is done in the Navigation frame when the view menu is changed. For example, after a successful login, the default console view is the organization view. When the view is changed to Users, the Navigation frame is redrawn to display all users; a search is performed to obtain this information. With a large number of users, disabling this search can drastically reduce the time it takes to load the Access Manager console. A filter can then be used to find the desired users. This option is available for any of the view menu types. To disable the search, add any of the following values: `orgs`, `orgUnits`, `users`, `policies`, `groups`, `roles`, `groupContainers`, and `peopleContainers`. If more than one value, they are comma-separated.

CAUTION The service attribute in the Access Manager console that corresponds to this property is Display Options, an organization attribute in the Administration Service. This console option takes precedence over any value defined in `com.iplanet.am.console.display.off`. If configuring this property in `AMConfig.properties`, do not configure it using the console (or vice versa).

- `com.iplanet.am.console.set.cn=true`

  If specified, the user common name (`cn`) will not be displayed in the Create User screen but it will be generated based on information entered in the First Name (`givenname`), Initial and Last Name (`sn`) fields of the User profile page and displayed as a read-only value on screen.

Cookies

These properties are specific to Access Manager cookies.

- `com.iplanet.am.cookie.name=iPlanetDirectoryPro`

  The value of this property is the name of the cookie. In an Access Manager deployment with more than one instance, it is recommended that the value of this property for one of the instances is changed.

NOTE The cookie name defined as `com.iplanet.am.cookie.name` is the Access Manager cookie and needs to be defined in a sticky load balancing situation. Do not use the HTTP session cookie as in some cases it is not retained.

- `com.iplanet.am.pcookie.name=DProPCookie`

  The value of this property is the name of the persistent cookie if that function is enabled.

- com.iplanet.am.cookie.secure=false

  This property allows the Access Manager cookie to be set in a secure mode in which the browser will only return the cookie when a secure protocol like HTTP(s) is used.

- com.iplanet.am.cookie.encode=COOKIE_ENCODE

  This property allows Access Manager to *URLencode* the cookie value which converts characters to ones that are understandable by HTTP.

Miscellaneous

This section is a catch-all for some miscellaneous and self-explanatory values.

- `com.iplanet.am.daemons=unix`
- `com.iplanet.am.locale=en_US`
- `com.iplanet.am.logstatus=ACTIVE`
- `com.iplanet.am.version=6.1`
- `com.iplanet.services.configpath=/etc/opt/SUNWam/config/ums`

  The value of this property is the path to the `serverconfig.xml` file. This file is discussed in Appendix B, "serverconfig.xml File."

Directory Server

This section describe the properties for the Directory Server data store.

Installation

These properties define the Directory Server to which the Access Manager points.

- `com.iplanet.am.directory.host=`*identity_server_host.domain_name*

The value of this property is the DNS domain name of the machine on which the Directory Server is located.

- `com.iplanet.am.directory.port=389`

  The value of this property is the port number of the machine on which the Directory Server is located. The default is `389`.

- `com.iplanet.am.server.protocol=http`

  The value of this property is the protocol used to communicate with the machine on which the Directory Server is located.

Directory Server Tree

The values of these properties are the top-level organization of the Directory Server tree defined during the installation process.

- `com.iplanet.am.defaultOrg=dc=`*top_level_org*`,dc=com`
- `com.iplanet.am.rootsuffix=dc=`*top_level_org*`,dc=com`
- `com.iplanet.am.domaincomponent=dc=`*top_level_org*`,dc=com`

Configuration Properties

There are a number of services configured in `AMConfig.properties` that can not be configured using the Access Manager console. These back-end services, and several attributes for other services, are defined in this section.

Debug Service

The Debug Service logs developer information in the case of application errors. (The Logging Service writes logs to be monitored by the application administrator.) More information on the Debug Service can be found in "Debug Files" on page 316 of Chapter 12, "Auditing Features."

- `com.iplanet.services.debug.level=error`

  The possible values for this property are: `off` | `error` | `warning` | `message`. They indicate the level of information recorded in the debug files.

- `com.iplanet.services.debug.directory=/var/opt/SUNWam/debug`

The value of this property is the output directory for the debug information. This directory should be writable by the server process.

NOTE In defining values for the Debug Service, remember that trailing spaces are significant. Also, on a Microsoft® Windows® system, use forward slashes "/" to separate directories.Finally, spaces in the file name are allowed only on a Windows system.

Stats Service

The following properties are used to configure the Stats Service for recording service statistics. This service is used by the Access Manager SDK and the Session Service. Code Example A-1 is a portion of the stats file to illustrate the information that is recorded. The file is named `amSDKStats` by default.

Code Example A-1 Portion of amSDKStats File

```
11/26/2002 01:46:18:592 PM PST: Thread[Thread-10,5,main]
SDK Cache Statistics
--------------------
Interval: 214
Hits during interval: 38
Hit ratio for this interval: 0.17757009345794392
Total number of requests: 214
Total number of Hits: 38
Overall Hit ratio: 0.17757009345794392
Total Cache Size: 72
```

- `com.iplanet.am.stats.interval=3600`

  The statistics interval should be at least 5 seconds to avoid CPU saturation. Access Manager will assume that any value less than that is 5 seconds.

- `com.iplanet.services.stats.directory=/var/opt/SUNWam/debug`

  This property specifies the output directory for the statistics files. By default, it is the same as the debug directory.

- `com.iplanet.services.stats.state=off`

Possible values for this directive are: `off | file | console`. `file` will write to a file named `amSDKStats` under the directory specified in the `com.iplanet.services.stats.directory` property and `console` will write into the deployment container log files.

NOTE In defining values for the Stats Service, remember that trailing spaces are significant. On a Windows system, use forward slashes "/" to separate directories. Spaces in the file name are also allowed on a Windows system.

Notification Service

The Notification Service allows Access Manager to send notifications to registered applications when an event has occurred (session destroyed, session timeout, etc.). This service also allows the single sign-on cache to stay up to date. The notification is basically a HTTP post message containing the *component notification* in its body.

- `com.iplanet.am.notification.url=`
  `http://`*identity_server_host.domain_name*`:`*port*`/amserver/notificationservice`

  The value of this property is the URI of the Notification Service.

When a notification task comes in, it is processed in the task queue. If it reaches the maximum length, further incoming requests will be rejected along with a ThreadPoolException, until the queue has vacancy

- `com.iplanet.am.notification.threadpool.size=10`

  This parameter is used to define the session thread pool for notification handling. It specifies the size of the pool as the total number of threads allowed.

- `com.iplanet.am.notification.threadpool.threshold= 100`

  This parameter specifies the maximum size of the task queue in the thread pool. A task is queued when no thread is available. If the number of unprocessed tasks reaches the value specified, no additional notification tasks will be accepted until there are vacancies. This value is dependent on the system memory resource; each task takes about 3k.

SDK Caching

The caching function in Access Manager is memory-based therefore when an identity-related object is created, deleted or modified, the cache is cleaned up. Each SDK cache entry stores a set of attributes and values of `AMObject` for a user. Because the size of each object is dependent upon the number of attributes it has, modifying these properties will affect the performance of Access Manager.

- `com.iplanet.am.sdk.cache.maxSize=10000`

  This property configures the size of the cache when caching is enabled. The value refers to the number of objects cached and should be an integer greater than 0; if not, the default 10000 will be used.

- `com.iplanet.am.session.maxSessions=5000`

  This property specifies the maximum number of concurrent sessions. Logging in when the maximum sessions has been met would send a Maximum Sessions error.

Online Certificate Status Protocol (OCSP)

OCSP is a protocol that specifies the syntax for communication between a server which holds certificate status and a client which is informed of said status.When a user attempts to access a server, OCSP sends a request for certificate status information and receives back a response of *current*, *expired* or *unknown*. If these properties are set, the certificate in question must be in the deployment container's certificate database. If the OCSP URL is set, the OCSP responder nickname must also be set or both will be ignored. If neither is set, the OCSP responder URL presented in the user's certificate will be used. If there is none in the user's certificate, no OCSP validation will be performed.

- `com.sun.identity.authentication.ocsp.responder.url`

  The value of this directive is the global OCSP responder URL for this instance of Access Manager, For example `http://ocsp.example.com/ocsp`.

- `com.sun.identity.authentication.ocsp.responder.nickname`

  The OCSP responder nickname refers to the Certificate Authority for the responder. This nickname is used to reference the Certificate Authority in the certificate itself.

Identity Object Processing

This property has a value equal to the implementation class of the module used for processing user creates, deletes, and modifies.

- `com.iplanet.am.sdk.userEntryProcessingImpl=`

Security

This property is used to enable Java security permissions. This permission is used to protect the Access Manager resources which should only be accessed by trusted resources. This permission is used to protect the admin DN and password as well as access to the encryption and decryption methods used to encrypt passwords. The default value is false. If enabled, modifications must be made to the deployed web container's Java policy file. This should be done as detailed in Code Example A-2.

Code Example A-2 Changes To Java Policy File

```
grant codeBase "file:{directory where jars are located}/-" {
      com.sun.identity.security.ISSecurityPermission "access",
                     "adminpassword,crypt";
 };
```

- `com.sun.identity.security.checkcaller=false`

SSL

This property is used to enable Secure Socket Layers (SSL). The default is `false`.

- `com.iplanet.am.directory.ssl.enabled=false`

Certificate Database

These properties are used by the command line utilities and SDK as well as the LDAP and Certificate-based authentication modules when initiating SSL connections to the Directory Server. It is also used when opening HTTP(S) connections from within the servlet container in the deployment container.

- `com.iplanet.am.admin.cli.certdb.dir=/`*IdentityServer_base*`/SUNWam/servers/alias`

  The value of this property is the name of the path to the certificate database.

- `com.iplanet.am.admin.cli.certdb.prefix=https-`*identity_server_host.domain_name-identity_server_host-*

  The value of this property is the certificate database prefix.

- `com.iplanet.am.admin.cli.certdb.passfile=/`*IdentityServer_base*`/SUNWam/config/.wtpass`

  The value of this property is the name of the file that contains the password for the certificate database.

NOTE When installing Access Manager, these values do not point to a configured certificate database. After creating the certificate database, these values should be reset to point to the Application Server as follows:

- `com.iplanet.am.admin.cli.certdb.dir=`/install_directory/SUNWappserver7/*domain*/*server_instance*/config
- `com.iplanet.am.admin.cli.certdb.prefix=`
- `com.iplanet.am.admin.cli.certdb.passfile=`/install_directory/SUNWappserver7/*domain*/*server_instance*/config/`.wtpass`

Access Manager should be restarted after the modifications.

Replication

These two properties are not required to support replication but they may be helpful in limiting errors due to latency. Enabling them may have a negative impact on performance but, if replication has significant latency, the retries may be enough to prevent Entry Not Found errors. For example, assume an Access Manager console is pointing to a read-only consumer configured to refer writes to a master. If a new organization is created, all write requests are referred to the master and then replicated back to the consumer. If Access Manager reads the organization back before it has been replicated to the consumer, it will get an Entry Not Found error.

NOTE It is not recommended to run the Access Manager console against a read-only consumer. The exception to this rule is when operating against user entries whose creations and modifications do not have the same latency problems as the SDK has special behavior to prevent such problems for these entries.

- `com.iplanet.am.replica.num.retries=0`

  This specifies the number of times to retry. When an Entry Not Found error is returned to the SDK, it will retry *n* times where *n* is the value of this property.

- `com.iplanet.am.replica.delay.between.retries=1000`

  This property specifies the delay time (in milliseconds) between the number of retries defined in the key above.

Event And LDAP Connection

These sets of properties are implemented when load balancers are used between the Identity SDK and the Directory Server. When the SDK performs an operation which fails, it will retry the operation as long as the exception is one defined in the `ldap.error.codes` property. These properties are necessary for failover configuration when it is accomplished via a load balancer as not all load balancers return the same error codes.

Event Connection

- `com.iplanet.am.event.connection.num.retries=3`

  This value specifies the number of time to retry an event connection.

- `com.iplanet.am.event.connection.delay.between.retries=3000`

  This value specifies the delay time (in milliseconds) between the number of retries defined in the key above.

- `com.iplanet.am.event.connection.ldap.error.codes.retries=80,81,91`

  This key specifies the `LDAPException` errors for which the retries will occur. The value is any valid LDAP error code.

LDAP Connection

The following keys are used to configure an LDAP connection for the add, delete modify, read and search methods.

- `com.iplanet.am.ldap.connection.num.retries=3`

  This value specifies the number of time to retry an LDAP connection.

- `com.iplanet.am.ldap.connection.delay.between.retries=1000`

  This value specifies the delay time (in milliseconds) between the number of retries defined in the key above.

- `com.iplanet.am.ldap.connection.ldap.error.codes.retries=80,81,91`

  This key specifies the `LDAPException` errors for which the retries will occur. The value is any valid LDAP error code.

SAML

These properties identify SAML-related configurations including properties relating to the Access Manager keystore file.

- `com.sun.identity.saml.removeassertion=false`

  This property indicates if assertions associated with artifacts and now de-referenced should be removed from the cache. If set to `true`, assertions will be removed. Otherwise, the assertion will be kept in memory and removed only when it is expired itself.

Keystore Properties

Each Access Manager has a keystore file used to store the certificates used for XML signing and verification. A stored certificate might include a partner site's certificate and the public key used by Access Manager to verify SAML responses and assertions from the partner. The keystore also holds the Access Manager certificate and the private key it uses to sign assertions. For more information on generating the keystore, certificate aliases and other functions, read about the `keytool`, a key and certificate management utility, in the `Readme.html` and `keystore.html` files located in the *IdentityServer_base*`/SUNWam/samples/saml/ xmlsig` directory.

- `com.sun.identity.saml.xmlsig.keystore=/`*IdentityServer_base*`/SUNWam/lib/keystore.jks`

  The value of this property is the name and location of the keystore file. Although, upon installing Access Manager, this property has a default value, the file itself is not initially generated and the name and location of the file can be changed.

- `com.sun.identity.saml.xmlsig.storepass=/`*IdentityServer_base*`/SUNWam/config/.storepass`

  The value of this property is the location of the password to the keystore.

- `com.sun.identity.saml.xmlsig.keypass=/`*IdentityServer_base*`/SUNWam/config/.keypass`

The value of this property is the location of the password to the private key which is used to sign the XML document.

- `com.sun.identity.saml.xmlsig.certalias=test`

  All entries (keys and trusted certificate entries) in the keystore file are accessed using unique aliases. The value of this property is the certificate alias of the Access Manager certificate which links to the private key used for signing assertions.

Miscellaneous Services

The following directives define the URIs for miscellaneous services.

- `com.iplanet.am.profile.host=`*identity_server_host.domain_name*

  The value of this property is the DNS domain name of the machine on which the Access Manager (and thus the Profile Service) is located.

- `com.iplanet.am.profile.port=58080`

  The value of this property is the port number used by the Access Manager (and thus the Profile Service). The default is 58080.

- `com.iplanet.am.naming.url=http://`*identity_server_host.domain_name*`:`*port*`/amserver/namingservice`

  The value of this property represents the URL where a request by the Access Manager or a remote single sign-on client will be sent to retrieve the URLs of Access Manager internal services. This is the URI for the Naming Service.

Read-Only Properties

The following properties are read-only and should not be modified. Any changes to these directives may render the Access Manager unusable.

Installation

These properties identify values defined during the installation process.

- `com.iplanet.am.installdir=/`*IdentityServer_base*`/SUNWam`

  This value is the base directory for the application.

- `com.iplanet.am.install.basedir=/`*IdentityServer_base*`/SUNWam/web-src/services/WEB-INF`

  This value is the base directory for the services.

- `com.iplanet.am.iASConfig=false`

  This property defines whether the Sun Java System Access Manager is running on the Sun Java System Application Server. The value is set during installation and must not be changed.

- `com.iplanet.am.console.remote=false`

  This property defines whether the console is installed on a remote or local machine. It is used by the Authentication Service and the console.

Deployment

These properties are used to identify the URIs for specific services and agents.

- `com.iplanet.am.services.deploymentDescriptor=/amserver`
- `com.iplanet.am.console.deploymentDescriptor=/amconsole`
- `com.iplanet.am.policy.agents.url.deploymentDescriptor=AGENT_DEPLOY_URI`

  This last property contains the name of the deployment container. Possible values here are BEA6.1, IBM 4.0.5, S1AS7.0, or WS.

- `com.sun.identity.webcontainer=WEB_CONTAINER`

NOTE Although the servlet and JSPs are deployment container independent, servlet 2.3 API `request.setCharacterEncoding()` (used to correctly decode incoming non-English characters) will not work if Access Manager is deployed on Sun Java System Web Server 6.0 or Sun Java System Application Server 7.0.

Shared Secret

This property is the shared secret for the Authentication Service.

- `com.iplanet.am.service.secret=AQIC5wM2LY4SfczLlj6134qMTx0nkE5XiFMg`

Session Properties

These properties are configurations for the Session Service.

- `com.iplanet.am.session.failover.enabled=false`

  This property is used to enable or disable the session failover feature. The following properties are used when this property is set to `true`.

  - `com.iplanet.am.localserver.protocol=http`
  - `com.iplanet.am.localserver.host=`*identity_server_host.domain_name*
  - `com.iplanet.am.localserver.port=58080`

- `com.iplanet.am.session.httpSession.enabled=true`

  When this property is set to true, an HttpSession will be created for the authenticated user in addition to an Access Manager session. This property is also related to session failover.

- `com.iplanet.am.session.invalidsessionmaxtime=3`

  This property disables a session if it is created and the user does not login before the time defined. The value is in minutes (for example, 3 minutes is the default value).

NOTE This value should always be greater than the time-out value in your authentication module properties file.

- `com.iplanet.am.session.client.polling.enable=false`

  If set to `true`, the client cache will invalidate itself after the amount of time defined in the next property, forcing data to reload.

- `com.iplanet.am.session.client.polling.period=180`

  This property defines the default polling period as 180 seconds.

- `com.sun.am.session.logging.enableHostLookUp=false`

  This property allows the session server (For example Access Manager) to look for the IP address from the host property and log it. If set to `true`, a reverse DNS lookup will be used to obtain the Domain Name from the IP address for logging purposes. If `false`, the IP address will be used thus, increasing performance.

- `com.iplanet.am.session.purgedelay=60`

This property defines the purge delay period in minutes. After a session times out, this is the extended time period for which the token will reside in the Session Service. This can be used by the client application to check if the session has timed out or not (using the SSO APIs). After this time period, the session is destroyed. The session token is in the INVALID state during this extended period.

NOTE	The session does not remain for this extended life if the user logs out or the session is explicitly destroyed by another Access Manager component.

- `com.iplanet.am.naming.failover.url=`

  This property can be used by any remote SDK application that wants failover in, for example, session validation or getting the service URLs.

Simple Mail Transfer Protocol (SMTP)

The following directives can be set to any valid SMTP server and port.

- `com.iplanet.am.smtphost=localhost`

NOTE	Because of how Microsoft® Windows 2000 processes this information, the default value of this directive, `localhost`, should be replaced by the actual mail server host name and the Access Manager should be restarted.

- `com.sun.identity.sm.smptpport=25`

Authentication

The following sections define properties used by the Authentication Service.

LDAP

- `com.iplanet.am.auth.ldap.createUserAttrList=`<*attr1,attr2,attr3...*>

  This property specifies a list of user attributes whose values will be retrieved from an external Directory Server during LDAP Authentication if the Authentication Service is configured for dynamically creating users. The new user created in the local Directory Server will have the values for these attributes retrieved from the external Directory Server.

SecurID

- `securidHelper.ports=58943`

  The value of this property is a space-separated list used by the SecurID Authentication module and helper(s).

Unix

- `unixHelper.port=58946`

  The value of this property is used in the Unix Authentication Service.

- `unixHelper.ipaddrs=`

  The value of this property can contain a list of trusted IP addresses. The IP addresses specified in this list are space-separated and will be read by the `amserver` script and passed to the Unix helper when starting it.

Security

Following are properties that define parameters for security purposes.

SecureRandom

This property specifies the factory class name for `SecureRandomFactory`.

- `com.iplanet.security.SecureRandomFactoryImpl=com.iplanet.am.util.JSSSecureRandomFactoryImpl`

  The available implementation classes are:

  a. `com.iplanet.am.util.JSSSecureRandomFactoryImpl` (uses JSS)

  b. `com.iplanet.am.util.SecureRandomFactoryImpl` (pure Java)

SocketFactory

This property specifies the factory class name for `LDAPSocketFactory`.

- `com.iplanet.security.SSLSocketFactoryImpl=com.iplanet.services.ldap.JSSSocketFactory`

  Available classes are:

  a. `com.iplanet.services.ldap.JSSSocketFactory` (uses JSS)

  b. `netscape.ldap.factory.JSSESocketFactory` (pure Java)

Encryption

These properties specify encryption information.

- `com.iplanet.security.encryptor=com.iplanet.services.util.JSSEncryption`

  The value specifies the encrypting class implementation. Available classes are:

  a. `com.iplanet.services.util.JCEEncryption`

  b. `com.iplanet.services.util.JSSEncryption.`

- `am.encryption.pwd=BcN2Vaek2TUcs3tvO7uW9bRIrcy/Koeo`

  This is the Data Encryption Standard (DES) encryption key password. The client needs this to decrypt the session ID for token creation. If decryption fails, the client will not be able to retrieve the protocol, server host and the server port information to construct the URL needed to search for a service. Do not change the value of this property without also re-encrypting the passwords in `serverconfig.xml`. For information, see Appendix B, "serverconfig.xml File."

NOTE When installing the Access Manager SDK remotely, the value of this property should be copied into the installation field labeled `The key used for encryption of passwords`. For information on how to install the Access Manager SDK remotely, see the Sun Java System Access Manager *Migration Guide*.

IP Address Checking

This property specifies whether the IP address of the client will be checked in `SSOToken` creations and validations.

- `com.iplanet.am.clientIPCheckEnabled=false`

Remote Policy API

These properties are defined for the Remote Policy API to use with policy agents.

- `com.sun.identity.agents.app.username=UrlAccessAgent`

  This property specifies the username for the Application authentication module.

- `com.sun.identity.agents.server.log.file.name=amRemotePolicyLog`

This property specifies the name of the file to use for logging remote policy messages. The directory where this file is located is defined in Logging Service settings.

- `com.sun.identity.agents.cache.size=1000`

  This property specifies the size of the cache created on the server where the policy agent resides.

- `com.sun.identity.agents.polling.interval=3`

  The polling interval is the duration of time for refreshing the cache

- `com.sun.identity.agents.notification.enabled=false`

  This property enable or disables notifications for remote policy API.

- `com.sun.identity.agents.notification.url=`

  This property defines the notification URL for remote policy API.

- `com.sun.identity.agents.logging.level=NONE`

  This property controls the granularity of logging for the remote policy API. The valid values are `ALLOW`, `DENY`, `BOTH` and `NONE`. The default value is `NONE`.

- `com.sun.identity.agents.use.wildcard=true`

  This property indicates whether to use wildcard for resource name comparison.

- `com.sun.identity.agents.header.attributes=cn,ou,o,mail,employeenumber,c`

  This property defines the attributes to be returned by policy evaluator. The specification is of the format *a*[,...] where *a* is the attribute in the data store that will be fetched.

- `com.sun.identity.agents.resource.comparator.class=com.sun.identity.policy.plugins.PrefixResourceName`

- `com.sun.identity.agents.resource.wildcard=*`

- `com.sun.identity.agents.resource.delimiter=/`

- `com.sun.identity.agents.resource.caseSensitive=false`

  This is to indicate whether case sensitivity is turned on or off during policy evaluation. The default value is `false` or off.

- `com.sun.identity.agents.true.value=allow`

This value is ignored if the application does not access the method `PolicyEvaluator.isAllowed`.

Policy

This property defines weights for policy subjects, rules and conditions. These weights influence the order in which these components are evaluated. The value is three integers delimited by "`:`". These integers indicate the proportional CPU cost for evaluating the three components, respectively.

- `com.sun.identity.policy.Policy.policy_evaluation_weights=10:10:10`

Federation

These properties configure information for the Federation Management module.

- `com.sun.identity.federation.fedCookieName=fedCookie`

  This property defines the name of the federation cookie.

- `com.sun.identity.federation.services.signingOn=false`

  This property defines whether federation requests and responses will be signed before sending. It also defines whether federation requests and responses that are received will be verified for signature validity. The default is `false`.

FQDN Map

The Fully Qualified Domain Name (FQDN) Map is a simple map that enables the Authentication Service to take corrective action in the case where a user may have typed in an incorrect URL either by specifying partial hostname or IP address to access a protected resource.

Valid values must comply with the syntax of this property which represent invalid FQDN values mapped to correct counterparts. The valid format for specifying these maps is:

`com.sun.identity.server.fqdnMap[`*invalid_name*`]=`*valid_name*

where *invalid_name* is a possible invalid FQDN host name that may be used by the user, and *valid_name* is the FQDN host name to which the filter will redirect the user.

CAUTION Ensure that there are no invalid or overlapping values for the same invalid FQDN name.

This property can also be used for creating a mapping for more than one host name. This may be the case when applications hosted on a server are accessible by more than one host name. It may also be used to configure Access Manager to NOT take corrective action for certain hostname URLs. For example, if no corrective action (such as a redirect) is desired for users who access application resources using a raw IP address, the map entry would look like:

`com.sun.identity.server.fqdnMap[`*IP_address*`]=`*IP_address*

Any number of values may be specified as long as they are valid and conform to the above stated requirements.

Examples of FQDN mapping might be:

- `com.sun.identity.server.fqdnMap[isserver]=isserver.mydomain.com`
- `com.sun.identity.server.fqdnMap[isserver.mydomain]=isserver.mydomain.com`
- `com.sun.identity.server.fqdnMap[`*IP_address*`]=isserver.mydomain.com`
- `com.sun.identity.server.fqdnMap[`*invalid_name*`]=`*valid_name*

Encryption Key

The value of this property is the password used to generate a symmetric key to encrypt and decrypt other sensitive data including the shared secret.

```
am.encryption.pwd=ro/LiN3pOxMXxtvbwf+owRFyzDYwxRTw
```

Appendix B

serverconfig.xml File

The file serverconfig.xml provides configuration information for Sun Java™ System Access Manager 6 2005Q1 regarding the Sun Java System Directory Server that is used as its data store. This chapter explains the elements of the file and how to configure it for failover, how can you have multiple instances, how can you undeploy the console and remove console files from a server. It contains the following sections:

- "Overview" on page 353
- "server-config Definition Type Document" on page 355
- "Failover Or Multimaster Configuration" on page 358

Overview

serverconfig.xml is located in */IdentityServer_base/*SUNWam/config/ums. It contains the parameters used by the Identity SDK to establish the LDAP connection pool to Directory Server. No other function of the product uses this file. Two users are defined in this file: user1 is a Directory Server proxy user and user2 is the Directory Server administrator.

Proxy User

The *Proxy User* can take on any user's privileges (for example, the organization administrator or an end user). The connection pool is created with connections bound to the proxy user. Access Manager creates a proxy user with the DN of cn=puser,ou=DSAME Users,dc=example,dc=com. This user is used for all queries made to Directory Server by IAccess Manager. It benefits from a proxy user ACI already configured in the Directory Server and, therefore, can perform actions on behalf of a user when necessary. It maintains an open connection through which all queries are passed (retrieval of service configurations, organization information, etc.). The proxy user password is always encrypted. Code Example B-1 illustrates where the encrypted password is located in `serverconfig.xml`.

Code Example B-1 Proxy User In serverconfig.xml

```
<User name="User1" type="proxy">
<DirDN>
cn=puser,ou=DSAME Users,dc=example,dc=com
</DirDN>
<DirPassword>
AQICkc3qIrCeZrpexyeoL4cdeXih4vv9aCZZ
</DirPassword>
</User>
```

Admin User

`dsameuser` is used for binding purposes when the Access Manager SDK performs operations on Directory Server that are not linked to a particular user (for example, retrieving service configuration information). Proxy User performs these operations on behalf of `dsameuser`, but a bind must first validate the `dsameuser` credentials. During installation, Access Manager creates `cn=dsameuser,ou=DSAME Users,dc=example,dc=com`. Code Example B-1 illustrates where the encrypted `dsameuser` password is found in `serverconfig.xml`.

Code Example B-2 Admin User In serverconfig.xml

```
<User name="User2" type="admin">
<DirDN>
cn=dsameuser,ou=DSAME Users,dc=example,dc=com
</DirDN>
<DirPassword>
AQICkc3qIrCeZrpexyeoL4cdeXih4vv9aCZZ
</DirPassword>
</User>
```

server-config Definition Type Document

`server-config.dtd` defines the structure for `serverconfig.xml`. It is located in *IdentityServer_base*/`SUNWam/dtd`. This section defines the main elements of the DTD. Code Example B-3 on page 357 is an example of the `serverconfig.xml` file.

iPlanetDataAccessLayer Element

iPlanetDataAccessLayer is the root element. It allows for the definition of multiple server groups per XML file. Its immediate sub-element is the ServerGroup Element. It contains no attributes.

ServerGroup Element

ServerGroup defines a pointer to one or more directory servers. They can be master servers or replica servers. The sub-elements that qualify the *ServerGroup* include Server Element, User Element, BaseDN Element and MiscConfig Element. The XML attributes of *ServerGroup* are the name of the server group, and *minConnPool* and *maxConnPool* which define the minimum (1) and maximum (10) connections that can be opened for the LDAP connection pool. More than one defined ServerGroup element is not supported.

NOTE Access Manager uses a connection pool to access Directory Server. All connections are opened when Access Manager starts and are not closed. They are reused.

Server Element

Server defines a specific Directory Server instance. It contains no sub-elements. The required XML attributes of *Server* are a user-friendly name for the server, the host name, the port number on which the Directory Server runs, and the type of LDAP connection that must be opened (either simple or SSL).

NOTE For an example of automatic failover using the Server element, see "Failover Or Multimaster Configuration" on page 358.

User Element

User contains sub-elements that define the user configured for the Directory Server instance. The sub-elements that qualify *User* include *DirDN* and *DirPassword*. It's required XML attributes are the name of the user, and the type of user. The values for *type* identify the user's privileges and the type of connection that will be opened to the Directory Server instance. Options include:

- auth—defines a user authenticated to Directory Server.
- proxy—defines a Directory Server proxy user. See "Proxy User" on page 353 for more information.
- rebind—defines a user with credentials that can be used to rebind.
- admin—defines a user with Directory Server administrative privileges. See "Admin User" on page 354 for more information.

DirDN Element

DirDN contains the LDAP Distinguished Name of the defined user.

DirPassword Element

DirPassword contains the defined user's encrypted password.

CAUTION It is important that passwords and encryption keys are kept consistent throughout the deployment. For example, the passwords defined in this element are also stored in Directory Server. If the password is to be changed in one place, it must be updated in both places. Additionally, this password is encrypted using the key defined in Appendix A, "AMConfig.properties File." If the encryption key defined in the `am.encryption.pwd` property is changed, all passwords in `serverconfig.xml` must be re-encrypted using `ampassword --encrypt` *password*. More information on this encryption utility can be found in the *Sun Java System Access Manager Administration Guide*.

BaseDN Element

BaseDN defines the base Distinguished Name for the server group. It contains no sub-elements and no XML attributes.

MiscConfig Element

MiscConfig is a placeholder for defining any LDAP JDK features like cache size. It contains no sub-elements. It's required XML attributes are the name of the feature and its defined value.

Code Example B-3 serverconfig.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!--
 Copyright (c) 2002 Sun Microsystems, Inc. All rights reserved.

 Use is subject to license terms.

-->
<iPlanetDataAccessLayer>
        <ServerGroup name="default" minConnPool="1" maxConnPool="10">
                <Server name="Server1" host="identity_server_host.domain_name"
port="389"
type="SIMPLE" />
                <User name="User1" type="proxy">
                        <DirDN>
                                cn=puser,ou=DSAME Users,dc=example,dc=com
                        </DirDN>
                        <DirPassword>
                                AQICkc3qIrCeZrpexyeoL4cdeXih4vv9aCZZ
                        </DirPassword>
                </User>
                <User name="User2" type="admin">
                        <DirDN>
                               cn=dsameuser,ou=DSAME Users,dc=example,dc=com
                        </DirDN>
                        <DirPassword>
                                AQICkc3qIrCeZrpexyeoL4cdeXih4vv9aCZZ
                        </DirPassword>
                </User>
                <BaseDN>
                        dc=example,dc=com
                </BaseDN>
        </ServerGroup>
</iPlanetDataAccessLayer>
```

Failover Or Multimaster Configuration

Access Manager allows automatic failover to any Directory Server defined as a Server Element in `serverconfig.xml`. More than one server can be configured for failover purposes or multimasters. If the first configured server goes down, the second configured server will takeover. Code Example B-4 illustrates `serverconfig.xml` with automatic failover configuration.

Code Example B-4 Configured Failover in serverconfig.xml

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!--
PROPRIETARY/CONFIDENTIAL. Use of this product is subject to license terms.
Copyright 2002 Sun Microsystems, Inc. All rights reserved.
-->
<iPlanetDataAccessLayer>
     <ServerGroup name="default" minConnPool="1" maxConnPool="10">
          <Server name="Server1" host="identity_server_host1.domain_name" port="389"
type="SIMPLE" />
          <Server name="Server2" host="identity_server_host2.domain_name" port="389"
type="SIMPLE" />
          <Server name="Server3" host="identity_server_host3.domain_name" port="390"
type="SIMPLE" />
          <User name="User1" type="proxy">
               <DirDN>
                    cn=puser,ou=DSAME Users,dc=example,dc=com
               </DirDN>
               <DirPassword>
                    AQIC5wM2LY4Sfcy+AQBQxghVwhBE92i78cqf
               </DirPassword>
          </User>
          <User name="User2" type="admin">
               <DirDN>
                    cn=dsameuser,ou=DSAME Users,dc=example,dc=com
               </DirDN>
               <DirPassword>
                    AQIC5wM2LY4Sfcy+AQBQxghVwhBE92i78cqf
               </DirPassword>
          </User>
          <BaseDN>
               o=isp
          </BaseDN>
     </ServerGroup>
</iPlanetDataAccessLayer>
```

Appendix C

WAR Files

Sun Java™ System Access Manager 6 2005Q1 contains a number of web application archive (WAR) files. These packages contain Java™ servlets and JavaServer Pages™ (JSP) pages that add functionality to the application. This chapter explains WAR files in general, their contents in an Access Manager deployment and which files can be modified. It contains the following sections:

Overview

The Java 2 Platform, Enterprise Edition (J2EE) platform (on which Access Manager is built) uses a component model to create full-scale applications. A *component* is self-contained functional software code assembled with other components into a J2EE application. The J2EE application components (which can be deployed separately on different servers) include:

1. Client components (including dynamic web pages, applets, and a Web browser) that run on the client machine.
2. Web components (including servlets and JSP) that run within a web container.
3. Business components (code that meets the needs of a particular enterprise domain such as banking, retail, or finance) that also run within the web container.
4. Enterprise infrastructure software that runs on legacy machines.

The *web components* tier in the Access Manager model can be customized based on each organization’s needs. This appendix concerns itself with this tier.

Web Components

When a web browser executes a J2EE application, it deploys server-side objects called *web components*. There are two types of web components: *Servlets* and *JavaServer Pages* (JSP).

- Servlets are small Java programs that dynamically process requests and construct responses from a web browser; they run within web containers.
- JSP are text-based documents that contain static template data [HTML, Scalable Vector Graphics (SVG), Wireless Markup Language (WML), or eXtensible Markup Language (XML)], and elements that construct dynamic content (in the case of Access Manager, servlets).

When a J2EE application is called, the JSP and corresponding servlets are constructed by the web browser.

Packaging Web Components

In general, all J2EE components are packaged separately and bundled together into an Enterprise Archive (EAR) file for application deployment. The Web Components, in particular, are packaged in *web application archives* (WAR). Each WAR contains the servlets and / or JSP, a deployment descriptor, and related resource files.

NOTE The WAR is the same format as a JavaARchive (JAR). However, an eXtensisible Markup Language (XML) deployment descriptor file must also be created.

Static HTML files and JSP are stored at the top level of the WAR directory. The top-level directory contains the WEB-INF sub-directory which contains the following:

- Server-side classes (Servlets, JavaBean components and related Java class files) must be stored in the `WEB-INF/classes` directory.
- Auxiliary JARs (tag libraries and any utility libraries called by server-side classes) must be stored in the `WEB-INF/lib` directory.

- web.xml—the web component deployment descriptor is stored in the WEB-INF directory.
- Tag library descriptor files

When modifying the files included in Access Manager WARs, customers are changing web components and thus, customizing their deployment.

NOTE	Be aware of any loss of the customized data during patch or upgrade.

WARs And Their Contents

Access Manager contains a number of WARs that can be modified to customize an Access Manager deployment. The WARs themselves are located in *IdentityServer_base*/`SUNWam` and include:

- `console.war`—files pertaining to the Access Manager console application.
- `password.war`—files pertaining to the Access Manager password reset service.
- `services.war`—contains files pertaining to Access Manager services.

The following sections detail the files within each WAR that can be modified and those that SHOULD NOT be modified.

console.war

The following sections detail the modifiable and non-modifiable documents contained within `console.war`. The path names are based on the directory structure discussed in Packaging Web Components.

console.war Modifiable Files

These directories contain files that can be modified.

- `web.xml` and related XML files used for constructructing it are located in *IdentityServer_base*/`SUNWam/web-src/applications/WEB-INF/`.
- Modifiable JavaScript files are located in *IdentityServer_base*/`SUNWam/web-src/applications/console/js/`.
- Modifiable JSP are located in the following directories dependant upon the service that deploys them:

    - *IdentityServer_base*/SUNWam/web-src/applications/console/auth/
    - *IdentityServer_base*/SUNWam/web-src/applications/console/federation/
    - *IdentityServer_base*/SUNWam/web-src/applications/console/policy/
    - *IdentityServer_base*/SUNWam/web-src/applications/console/service/
    - *IdentityServer_base*/SUNWam/web-src/applications/console/session/
    - *IdentityServer_base*/SUNWam/web-src/applications/console/user/
- Modifiable image files are located in *IdentityServer_base*/SUNWam/web-src/applications/console/images/.
- Modifiable stylesheets are located in *IdentityServer_base*/SUNWam/web-src/applications/console/css/.

console.war Non-Modifiable Files

These directories contain files that SHOULD NOT be modified.

- JARs are located in *IdentityServer_base*/SUNWam/web-src/applications/WEB-INF/lib/.
- Tag Library Descriptor (.tld) files are located in *IdentityServer_base*/SUNWam/web-src/applications/WEB-INF/.

password.war

The following sections detail the modifiable and non-modifiable documents contained within password.war. The path names are based on the directory structure discussed in Packaging Web Components.

password.war Modifiable Files

These directories contain files that can be modified.

- web.xml and related XML files used for constructructing it are located in *IdentityServer_base*/SUNWam/web-src/password/WEB-INF/.
- Modifiable JSP are located in *IdentityServer_base*/SUNWam/web-src/password/password/ui/.
- Modifiable image files are located in *IdentityServer_base*/SUNWam/web-src/password/password/images/.
- Modifiable stylesheets are located in *IdentityServer_base*/SUNWam/web-src/password/password/css/.

password.war Non-Modifiable Files

These directories contain files that SHOULD NOT be modified.

- Non-modifiable JARs are located in *IdentityServer_base*/`SUNWam/web-src/password/WEB-INF/lib/`.
- Non-modifiable tag library descriptor (.tld) files are located in *IdentityServer_base*/`SUNWam/web-src/password/WEB-INF/`.

services.war

The following sections detail the modifiable and non-modifiable documents contained within `services.war`. The path names are based on the directory structure discussed in Packaging Web Components.

services.war Modifiable Files

These directories contain files that can be modified.

- `web.xml` and related XML files used for constructructing it are located in *IdentityServer_base*/`SUNWam/web-src/services/WEB-INF/`.
- JavaScript files are located in *IdentityServer_base*/`SUNWam/web-src/services/js/`.
- JSP are located in the following directories dependant upon the service that requires the customization:
  - *IdentityServer_base*/`SUNWam/web-src/services/config/auth/default/`
  - *IdentityServer_base*/`SUNWam/web-src/services/config/federation/default/`
- Image files are located in the following directories dependant upon the service to which the images apply:
  - *IdentityServer_base*/`SUNWam/web-src/services/images/`
  - *IdentityServer_base*/`SUNWam/web-src/services/fed_images/`
  - *IdentityServer_base*/`SUNWam/web-src/services/login_images/`
- Stylesheets are located in the following directories dependant upon the service to which they apply:
  - *IdentityServer_base*/`SUNWam/web-src/services/css/`.
  - *IdentityServer_base*/`SUNWam/web-src/services/fed_css/`.

services.war Non-Modifiable Files

These directories contain files that SHOULD NOT be modified.

- Non-modifiable JARs are located in *IdentityServer_base*/`SUNWam/web-src/services/WEB-INF/lib/`.
- Non-modifiable Tag Library Descriptor (.tld) files are located in *IdentityServer_base*/`SUNWam/web-src/services/WEB-INF/`.

Updating Modified WARs

Once a file within a WAR is modified, the WAR itself needs to be updated with the newly modified file. Following is the procedure to update a WAR.

1. cd *IdentityServer_base* / `SUNWam`

   This is the directory in which the WARs are kept.

2. jar **-**uvf *WARfilename*.war <*path_to_modified_file*>

   The `-uvf` option replaces the old file with the newly modified file. For example:

   ```
   jar -uvf console.war newfile/index.html
   ```

   replaces the `index.html` file in console.war with the `index.html` file located in *IdentityServer_base*/`SUNWam/newfile`.

3. `rm newfile/index.html`

   Delete the modified file.

Redeploying Modified WARs

Once updated, the WARs need be redeployed to their web container. The web container provides services such as request dispatching, security, concurrency, and life cycle management. It also gives the web components access to the J2EE APIs. The following procedures are specific to each particular WAR and web container. After redeploying the war files, all related servers need to be restarted.

NOTE The BEA WebLogic Server 6.1 and Sun Java System Application Server web containers do not require WARs to be exploded. They are deployed as WARs.

BEA WebLogic Server 6.1

The following commands are used on BEA WebLogic Server 6.1 to redeploy Access Manager WARs.

NOTE `amconsole`, `amserver` and `ampassword` are the default console, server and password deploy URIs, respectively.

To Deploy console.war On WebLogic

java weblogic.deploy -url *protocol://server_host:server_port* -component amconsole:*WL61 _server_name* deploy *WL61_admin_password* amconsole *IdentityServer_base* / `SUNWam` / console.war

To Deploy services.war on WebLogic

java weblogic.deploy -url *protocol://server_host:server_port* -component amserver:*WL61 _server_name* deploy *WL61_admin_password* amserver *IdentityServer_base* / `SUNWam` / services.war

To Deploy password.war on WebLogic

java weblogic.deploy -url *protocol://server_host:server_port* -component ampassword:*WL61 _server_name* deploy *WL61_admin_password* ampassword *IdentityServer_base* / `SUNWam` / password.war

NOTE For more complete information on the Java utility `weblogic.deploy` and its options, see the BEA WebLogic Server 6.1 documentation.

Sun Java System Application Server 7.0

The following commands are used on Sun Java System Application Server 7.0 to redeploy Access Manager WARs.

To Deploy console.war On Sun Java System Application Server

asadmin deploy -u *S1AS_administrator* -w *S1AS_administrator_password* -H *console_server_host* -p *S1AS_server_port* --type web *secure_flag* --contextroot *console_deploy_uri* --name `amconsole` --instance *S1AS_instance* *IdentityServer_base* / `SUNWam` / console.war

To Deploy services.war On Sun Java System Application Server

asadmin deploy -u *S1AS_administrator* -w *S1AS_administrator_password* -H *server_host* -p *S1AS_server_port* --type web *secure_flag* --contextroot *server_deploy_uri* --name amserver --instance *S1AS_instance* *IdentityServer_base*/`SUNWam`/services.war

To Deploy password.war on Sun Java System Application Server

asadmin deploy -u *S1AS_administrator* -w *S1AS_administrator_password* -H *console_server_host* -p *S1AS_administrator_server_port* --type web *secure_flag* --contextroot *password_deploy_uri* --name ampassword --instance *S1AS_instance* *IdentityServer_base*/`SUNWam`/password.war

NOTE For more complete information on the `asadmin` deploy command and its options, see the Sun Java System Application Server 7.0 Developer's Guide.

IBM WebSphere Application Server

For detailed instructions on how to deploy WARs in an IBM WebSphere Application Server container, see the documentation at `http://www-3.ibm.com/software/webservers/studio/doc/v40/studioguide/en/html/sdsscenario1.html`.

Appendix D

Notification Service

Sun Java™ System Access Manager 6 2005Q1 Notification Service allows for session notifications to be sent to remote web containers. It is necessary to enable this service for use by SDK applications running remotely from the Access Manager server itself. This chapter explains how to enable a remote web container to receive the notifications. It contains the following sections:

Overview

The Notification Service allows for session notifications to be sent to web containers that are running the Access Manager SDK remotely. The notifications apply to the Session, Policy and Naming Services only. In addition, the remote application must be running in a web container. The purpose of the notifications would be:

- To sync up the client side cache of the respective services.
- To enable more real time updates on the clients. (Polling is used in absence of notifications.)
- No client application changes are required to support notifications.

Note that the notifications can be received only if the remote SDK is installed on a web container.

Enabling The Notification Service

Following are the steps to configure the remote SSO SDK to receive session notifications. Setting up clients to receive notifications

1. Install Access Manager on Machine 1.
2. Install Sun Java System Web Server on Machine 2.
3. Install the `SUNWamsdk` on the same machine as the Web Server.

   For instructions on installing the Access Manager SDK remotely, see the *Sun Java™ Enterprise System 2005Q1 Installation Guide*.
4. Ensure that the following are true concerning the machine where the SDK is installed.

   a. Ensure that the right access permissions are set for the /*remote_SDK_server*/`SUNWam/lib` and /*remote_SDK_server*/`SUNWam/locale` directories on the server where the SDK is installed.

   These directories contains the files and jars on the remote server.

   b. Ensure that the following permissions are set in the Grant section of the `server.policy` file of the Web Server.

   `server.policy` is in the `config` directory of the Web Server installation. These permissions can be copied and pasted, if necessary:

   ```
   permission java.security.SecurityPermission
   "putProviderProperty.Mozilla-JSS"
   ```

   ```
   permission java.security.SecurityPermission
   "insertProvider.Mozilla-JSS";
   ```

   c. Ensure that the correct classpath is set in `server.xml`.

   `server.xml` is also in the `config` directory of the Web Server installation. A typical classpath would be:

   ```
   <JAVA javahome="/export/home/ws61/bin/https/jdk"
   serverclasspath="/export/home/ws61/bin/https/jar/webserv-rt.jar:
   ${java.home}/lib/tools.jar:/export/home/ws61/bin/https/jar/webse
   rv-ext.jar:/export/home/ws61/bin/https/jar/webserv-jstl.jar:/exp
   ort/home/ws61/bin/https/jar/nova.jar"
   classpathsuffix="::/IS_CLASSPATH_BEGIN_DELIM://usr/share/lib/xal
   an.jar:/export/SUNWam/lib/xmlsec.jar://usr/share/lib/xercesImpl.
   jar://usr/share/lib/sax.jar://usr/share/lib/dom.jar:/export/SUNW
   am/lib/dom4j.jar:/export/SUNWam/lib/jakarta-log4j-1.2.6.jar:/usr
   ```

```
/share/lib/jaxm-api.jar:/usr/share/lib/saaj-api.jar://usr/share/
lib/jaxrpc-api.jar://usr/share/lib/jaxrpc-impl.jar:/export/SUNWa
m/lib/jaxm-runtime.jar:/usr/share/lib/saaj-impl.jar:/export/SUNW
am/lib:/export/SUNWam/locale://usr/share/lib/mps/jss3.jar:/expor
t/SUNWam/lib/am_sdk.jar:/export/SUNWam/lib/am_services.jar:/expo
rt/SUNWam/lib/am_sso_provider.jar:/export/SUNWam/lib/swec.jar:/e
xport/SUNWam/lib/acmecrypt.jar:/export/SUNWam/lib/iaik_ssl.jar:/
/usr/share/lib/jaxp-api.jar://usr/share/lib/mail.jar://usr/share
/lib/activation.jar:/export/SUNWam/lib/servlet.jar:/export/SUNWa
m/lib/am_logging.jar:/usr/share/lib/commons-logging.jar:/IS_CLAS
SPATH_END_DELIM:" envclasspathignored="true" debug="false"
debugoptions="-Xdebug
-Xrunjdwp:transport=dt_socket,server=y,suspend=n"
javacoptions="-g" dynamicreloadinterval="2">
```

5. Use the SSO samples installed on the remote SDK server for configuration purposes.

   a. Change to the /*remote_SDK_server*/`SUNWam/samples/sso` directory.

   b. Run `gmake`.

   c. Copy the generated class files from /*remote_SDK_server*/`SUNWam/samples/sso` to /*remote_SDK_server*/`SUNWam/lib/`.

6. Copy the encryption value of `am.encryption.pwd` from the `AMConfig.properties` file installed with Access Manager to the `AMConfig.properties` file on the remote server to which the SDK was installed.

   The value of `am.encryption.pwd` is used for encrypting and decrypting passwords.

7. Login into Access Manager as `amadmin`.

   `http://`*AccessManager_host*`:3000/amconsole`

8. Execute the servlet by entering `http://`*remote_SDK_host*`:58080/servlet/SSOTokenSampleServlet` into the browser location field and validating the `SSOToken`.

   `SSOTokenSampleServlet` is used for validating a session token and adding a listener. Executing the servlet will print out the following message:

   ```
   SSOToken host name: 192.18.149.33 SSOToken Principal name:
   uid=amAdmin,ou=People,dc=red,dc=iplanet,dc=com Authentication type
   used: LDAP IPAddress of the host: 192.18.149.33 The token id is
   AQIC5wM2LY4SfcyURnObg7vEgdkb+32T43+RZN30Req/BGE= Property: Company
   is - Sun Microsystems Property: Country is - USA SSO Token
   Validation test Succeeded
   ```

9. Set the property `com.iplanet.am.notification.url=` in `AMConfig.properties` of the machine where the Client SDK is installed:

   ```
   com.iplanet.am.notification.url=http://clientSDK_host.domain:port/servlet
       com.iplanet.services.comm.client.PLLNotificationServlet
   ```

10. Restart the Web Server.

11. Login into Access Manager as `amadmin`.

    `http://`*AccessManager_host*`:3000/amconsole`

12. Execute the servlet by entering `http://`*remote_SDK_host*`:58080/servlet/SSOTokenSampleServlet` into the browser location field and validating the `SSOToken` again.

    When the machine on which the remote SDK is running receives the notification, it will call the respective listener when the session state is changed. Note that the notifications can be received only if the remote SDK is installed on a web container.

Appendix E

Directory Server Concepts

Sun Java™ System Access Manager 6 2005Q1 uses Sun Java System Directory Server to store its data. Certain features of the LDAP-based Directory Server are used by Access Manager to help manage its data. This chapter contains information on these Directory Server features and how they are used. It contains the following sections:

Overview

Because Access Manager needs an underlying data store, it has been built to work with Sun Java System Directory Server. They are complementary in architecture and design data. Use of Directory Server, though, may not be exclusive to Access Manager and therefore, needs to be treated as a completely separate deployment. For more information on Directory Server deployment, see the Sun Java System Directory Server documentation.

This appendix explains three Directory Server functions that are used by the Access Manager. A *role* is an identity grouping mechanism; an *access control instruction* (ACI) defines rules to allow or deny access to Directory Server data, and *class of service* is an attribute grouping mechanism.

Roles

Roles are a Directory Server entry mechanism similar to the concept of a *group*. A group has members; a role has members. A role's members are LDAP entries that are said to *possess* the role. The criteria of the role itself is defined as an LDAP entry with attributes, identified by the Distinguished Name (DN) attribute of the entry. Directory Server has a number of different types of roles but Access Manager can only manage one of them: the managed role.

NOTE The other Directory Server role types can still be used in a directory deployment; they just can not be managed by Access Manager.

Users can possess one or more roles. For example, a contractor role which has attributes from the Session Service and the URL Policy Agent Service might be created. Thus, when new contractors start, the administrator can assign them this role rather than setting separate attributes in the contractor entry. If the contractor were then to become a full-time employee, the administrator would just re-assign the user a different role.

Managed Roles

With a managed role, membership is defined in each member entry and not in the role definition entry. An attribute which designates membership is placed in each LDAP entry that possesses the role. This is in sharp contrast to a traditional static group which centrally lists the members in the group object entry itself.

NOTE By inverting the membership mechanism, the role will scale better than a static group. In addition, the referential integrity of the role is simplified, and the roles of an entry can be easily determined.

An administrator assigns the role to a member entry by adding the `nsRoleDN` attribute to it. The value of `nsRoleDN` is the DN of the role definition entry. The following apply to managed roles:

- Multiple managed roles can be created for each organization or sub-organization.
- A managed role can be enabled with any number of services.
- Any user that possesses a role with a service will inherit the service attributes from that role.

NOTE All Access Manager roles can only be configured directly under organization or sub-organization entries.

Definition Entry

A role's definition entry is a LDAP entry in which the role's characteristic attributes are defined. These attributes are passed onto the member entry. Below is a sample LDAP entry that represents the definition entry of a manager role.

Code Example 14-1 LDAP Definition Entry

```
dn: cn=managerrole,dc=siroe,dc=com
     objectclass: top
     objectclass: LDAPsubentry
     objectclass: nsRoleDefinition
     objectclass: nsSimpleRoleDefinition
     objectclass: nsManagedRoleDefinition
     cn: managerrole
     description: manager role within company
```

The `nsManagedRoleDefinition` object class inherits from the `LDAPsubentry`, `nsRoleDefinition` and `nsSimpleRoleDefinition` object classes.

Member Entry

A role's member entry is a LDAP entry to which the role is applied. An LDAP entry that contains the attribute `nsRoleDN` and its value DN indicates that the entry has the characteristics defined in the value DN entry. In Code Example 14-2 below, the DN identifies Code Example 14-1 above as the role definition entry: `cn=managerrole,dc=siroe,dc=com`.

Virtual Attribute

When a member entry that contains the `nsRoleDN` attribute is returned by a Directory Server search, `nsRoleDN` will be duplicated as the `nsRole` attribute in the same entry. `nsRole` will carry a value of any managed, filtered or nested roles assigned to the user (such as `ContainerDefaultTemplateRole`). Code Example 14-2 on page 374 includes this virtual attribute when returned by Directory Server only.

Code Example 14-2 LDAP Member Entry

```
dn: uid=managerperson,ou=people,dc=siroe,dc=com
     objectclass: top
     objectclass: person
     objectclass: inetorgperson
     uid: managerperson
     gn: manager
     sn: person
     nsRoleDN: cn=managerrole,ou=people,dc=siroe,dc=com
     nsRole: cn=managerrole,ou=people,dc=siroe,dc=com
     nsRole: cn=containerdefaulttemplaterole,ou=people,dc=siroe,dc=com
     description: manager person within company
```

How Access Manager Uses Roles

Access Manager uses roles to apply Access Control Instructions. When first installed, the Access Manager configures ACI that define administrator permissions to directory data. These ACI are then designated in roles (such as `Organization Admin Role` and `Organization Help Desk Admin Role`) which, when assigned to a user, define the user's level of access. For a list of roles created for each Access Manager object configured, see "Access Control Instructions" on page 376.

NOTE Managed groups in Access Manager are modeled almost the same as roles. They add an attribute to an LDAP entry to make the entry a member of the dynamic group

Role Creation

When a role is created, it contains the auxiliary LDAP object class `iplanet-am-managed-role`. This object class, in turn, contains the following allowed attributes:

- `iplanet-am-role-managed-container-dn` contains the DN of the identity-related object that the role was created to manage.
- `iplanet-am-role-type` contains a value used by the Access Manager console for display purposes. After authentication, the console gets the user's roles and checks this attribute for the correct page to display based on which of the following three values it has:
    - 1 for top-level administrator only.

- 2 for all other administrators.
- 3 for user.

If the user has no administrator roles, the User profile page will display. If the user has an administrator role, the console will start the user at the top-most administrator page based on which value is present.

NOTE When Access Manager attempts to process two templates that are set to the same priority level, Directory Server arbitrarily picks one of the templates to return. For more information, see the Sun Java System Directory Server documentation.

Role Location

All roles in an organization are viewed from the organization's top-level. For example, if an administrator wants to add a user to the administrator role for a people container, the administrator would go to the organization above the people container, look for the role based on the people container's name, and add the user to the role.

NOTE Alternately, an administrator might go to the user profile and add the role to the user.

Displaying The Correct Login Start Page

The attribute `iplanet-am-user-admin-start-dn` can be defined for a role or a user; it would override the `iplanet-am-role-type` attribute by defining an alternate display page URL. Upon a user's successful authentication:

1. Access Manager checks the `iplanet-am-user-admin-start-dn` for the user.

   This attribute is contained in the User service. If it is set, the user is started at this point. If not, Access Manager goes to step 2.

NOTE The value of `iplanet-am-user-admin-start-dn` can override the administrator's start page. For example, if a group administrator has read access to the top-level organization, the default starting page of the top-level organization, taken from `iplanet-am-role-type`, can be overridden by defining `iplanet-am-user-admin-start-dn` to display the group's start page.

2. Access Manager checks the user for the value of `iplanet-am-role-type`.

   If the attribute defines an administrator-type role, the value of `iplanet-am-role-managed-container-dn` is retrieved and the highest point in the directory tree is displayed as a starting point. For more information on the `iplanet-am-role-type` attribute, see "Role Creation" on page 374.

NOTE If the attribute has no value, a search from Access Manager root is performed for all container-type objects; the highest object in the directory tree that corresponds to the `iplanet-am-role-type` value is where the user starts. Although rare, this step is memory-intensive in very large directory trees with many container entries.

Access Control Instructions

Control over access to directory information is implemented in Access Manager using roles. Users inherit access permissions based on their role membership and parent organization. Access Manager installs pre-configured administrator roles that define different levels of permission for administrators to access directory information; these roles are dynamically created when a group, organization, container or people container object is configured. They are:

- `Organization Admin`
- `Organization Help Desk Admin`
- `Group Admin`
- `Container Admin`
- `Container Help Desk Admin`
- `People Container Admin`.

NOTE This section refers to ACIs as they are applied to administrative roles only. Policy is another form of access control which are created and used in Access Manager but apply to web resources not Directory Server data.

These default roles, when possessed by a user entry, define that user's level of access to Directory Server data. For example, when an organization is created, the Access Manager SDK creates an `Organization Admin` role and an `Organization Help Desk Admin` role. The permissions are read and write access to all entries in the organization and read access to all entries in the organization, respectively.

NOTE The Access Manager SDK gets the ACIs from the attribute `iplanet-am-admin-console-dynamic-aci-list` (defined in the `amAdminConsole.xml` service file) and sets them in the roles after they have been created.

Defining ACIs

ACIs are defined in the Access Manager console administration XML service file, `amAdminConsole.xml`. This file contains two global attributes that define ACIs for use in Access Manager: `iplanet-am-admin-console-role-default-acis` and `iplanet-am-admin-console-dynamic-aci-list`.

iplanet-am-admin-console-role-default-acis

This global attribute defines which *Access Permissions* are displayed in the Create Role screen of the Access Manager console. By default, `Organization Admin`, `Organization Help Desk Admin` and `No Permissions` are displayed. If other default permissions are desired, they must be added to this attribute.

iplanet-am-admin-console-dynamic-aci-list

This global attribute is where all of the defined administrator-type ACIs are stored. For information on how ACIs are structured, see "Format of Predefined ACIs" on page 377.

NOTE Because ACIs are stored in the role, changing the default permissions in `iplanet-am-admin-console-dynamic-aci-list` after a role has been created will not affect it. Only roles created after the modification has been made will be affected.

Format of Predefined ACIs

ACIs defined in Access Manager for use with administrator-type roles follow a different format than those defined using Directory Server. The format of the predefined Access Manager ACI is `permissionName | ACI Description | DN:ACI ## DN:ACI ## DN:ACI` where:

- `permissionName`—The name of the permission which generally includes the object being controlled and the type of access. For example, `Organization Admin` is an administrator that controls access to an organization object.

- `ACI Description`—A text description of the access the ACI allows.
- `DN:ACI`—There can be any number of `DN:ACI` pairs separated by the ## symbols. The SDK will get and set each pair in the entry named by `DN`. This format also supports tags which can be dynamically substituted when the role is created. Without these tags, the DN and ACI would be hard-coded to specific organizations in the directory tree which would make them unusable as defaults. For example, if there is a default set of ACIs for every `Organization Admin`, the organization name should not be hard-coded in this role. The supported tags are `ROLENAME`, `ORGANIZATION`, `GROUPNAME`, and `PCNAME`. These tags are substituted with the DN of the entry when the corresponding entry type is created. See the "Default ACIs" on page 378 for examples of ACI formats. Additionally, more complete ACI information can be found in the Sun Java System Directory Server documentation.

NOTE If there are duplicate ACI within the default permissions, the SDK will print a debug message.

Default ACIs

Following are the default ACIs installed by Access Manager. They are copied from a Access Manager configuration whose top-level organization is configured as `o=isp`.

- `Top Level Admin|Access to all entries|o=isp:aci: (target="ldap:///o=isp")(targetattr="*")(version 3.0; acl "Proxy user rights"; allow (all) roledn = "ldap:///ROLENAME";)`
- `Organization Admin|Read and Write access to all organization entries|o=isp:aci:(target="ldap:///($dn),o=isp")(targetfilter=(!(|(nsroledn=cn=Top Level Admin Role,o=isp)(nsroledn=cn=Top Level Help Desk Admin Role,o=isp))))(targetattr = "*")(version 3.0; acl "Organization Admin Role access allow"; allow (all) roledn = "ldap:///cn=Organization Admin Role,[$dn],o=isp";)##o=isp:aci: (target="ldap:///cn=Organization Admin Role,($dn),o=isp")(targetattr="*")(version 3.0; acl "Organization Admin Role access deny"; deny (write,add,delete,compare,proxy) roledn = "ldap:///cn=Organization Admin Role,($dn),o=isp";)`
- `Organization Help Desk Admin|Read access to all organization entries|ORGANIZATION:aci:(target="ldap:///ORGANIZATION")(targetfilter=(!(|(nsroledn=cn=Top Level Admin Role,o=isp)(nsroledn=cn=Top Level Help Desk Admin Role,o=isp)(nsroledn=cn=Organization Admin Role,ORGANIZATION))))(targetattr = "*") (version 3.0; acl`

"Organization Help Desk Admin Role access allow"; allow (read,search) roledn = "ldap:///ROLENAME";)##ORGANIZATION:aci: (target="ldap:///ORGANIZATION")(targetfilter=(!(|(nsroledn=cn=Top Level Admin Role,o=isp)(nsroledn=cn=Organization Admin Role,ORGANIZATION))))(targetattr = "userPassword") (version 3.0; acl "Organization Help Desk Admin Role access allow"; allow (write)roledn = "ldap:///ROLENAME";)

- Container Admin|Read and Write access to all organizational unit entries|o=isp:aci:(target="ldap:///($dn),o=isp")(targetfilter=(!(|(nsroledn=cn=Top Level Admin Role,o=isp)(nsroledn=cn=Top Level Help Desk Admin Role,o=isp))))(targetattr = "*")(version 3.0; acl "Container Admin Role access allow"; allow (all) roledn = "ldap:///cn=Container Admin Role,[$dn],o=isp";)o=isp:aci: (target="ldap:///cn=Container Admin Role,($dn),o=isp")(targetattr="*")(version 3.0; acl "Container Admin Role access deny"; deny (write,add,delete,compare,proxy) roledn = "ldap:///cn=Container Admin Role,($dn),o=isp";)

- Container Help Desk Admin|Read access to all organizational unit entries|ORGANIZATION:aci:(target="ldap:///ORGANIZATION")(targetfilter=(!(|(nsroledn=cn=Top Level Admin Role,o=isp)(nsroledn=cn=Top Level Help Desk Admin Role,o=isp)(nsroledn=cn=Container Admin Role,ORGANIZATION))))(targetattr = "*") (version 3.0; acl "Container Help Desk Admin Role access allow"; allow (read,search) roledn = "ldap:///ROLENAME";)##ORGANIZATION:aci: (target="ldap:///ORGANIZATION")(targetfilter=(!(|(nsroledn=cn=Top Level Admin Role,o=isp)(nsroledn=cn=Container Admin Role,ORGANIZATION))))(targetattr = "userPassword") (version 3.0; acl "Container Help Desk Admin Role access allow"; allow (write) roledn = "ldap:///ROLENAME";)

- Group Admin|Read and Write access to all group members|ORGANIZATION:aci:(target="ldap:///GROUPNAME")(targetattr = "*") (version 3.0; acl "Group and people container admin role"; allow (all) roledn = "ldap:///ROLENAME";)##ORGANIZATION:aci: (target="ldap:///ORGANIZATION")(targetfilter=(!(|(!FILTER)(|(nsroledn=cn=Top Level Admin Role,o=isp)(nsroledn=cn=Top Level Help Desk Admin Role,o=isp)(nsroledn=cn=Organization Admin Role,ORGANIZATION)(nsroledn=cn=Container Admin Role,ORGANIZATION)))))(targetattr != "iplanet-am-web-agent-access-allow-list || iplanet-am-web-agent-access-not-enforced-list ||

```
iplanet-am-domain-url-access-allow ||
iplanet-am-web-agent-access-deny-list")(version 3.0;acl "Group
admin's right to the members"; allow (read,write,search) roledn =
"ldap:///ROLENAME";)
```

- `People Container Admin|Read and Write access to all users|ORGANIZATION:aci:(target="ldap:///PCNAME")(targetfilter=(!(|(nsroledn=cn=Top Level Admin Role,o=isp)(nsroledn=cn=Top Level Help Desk Admin Role,o=isp)(nsroledn=cn=Organization Admin Role,ORGANIZATION)(nsroledn=cn=Container Admin Role,ORGANIZATION))))(targetattr != "iplanet-am-web-agent-access-allow-list || iplanet-am-web-agent-access-not-enforced-list || iplanet-am-domain-url-access-allow || iplanet-am-web-agent-access-deny-list") (version 3.0; acl "People container admin role"; allow (all) roledn = "ldap:///ROLENAME";)`

NOTE Access Manager generates a `Top Level Admin` and `Top Level Help Desk Admin` during installation. These roles can not be dynamically generated for any other identity-type objects but the top-level organization.

Class Of Service

Both dynamic and policy attributes use *class of service* (CoS), a feature of the Directory Server that allows attributes to be created and managed in a single central location, and dynamically added to user entries as the user entry is called. Attribute values are not stored within the entry itself; they are generated by CoS as the entry is sent to the client browser. Dynamic and policy attributes using CoS consist of the following two LDAP entries:

- CoS Definition Entry—This entry identifies the type of CoS being used (Classic CoS). It contains all the information, except the attribute values, needed to generate an entry defined with CoS. The scope of the CoS is the entire sub-tree below the parent of the CoS definition entry.
- Template Entry—This entry contains a list of the attribute values that are generated when the target entry is displayed. Changes to the attribute values in the Template Entry are automatically applied to all entries within the scope of the CoS.

The CoS Definition entry and the Template entry interact to provide attribute information to their target entries; any entry within the scope of the CoS. Only those services which have dynamic or policy attributes use the Directory Server CoS feature; no other services do.

NOTE For additional information on the CoS feature, see the Sun Java System Directory Server documentation.

CoS Definition Entry

CoS definition entries are stored as LDAP subentries under the organization level but can be located anywhere in the DIT. They contain the attributes specific to the type of CoS being defined. These attributes name the *virtual* CoS attribute, the template DN and, if necessary, the specifier attribute in target entries. By default, the CoS mechanism will not override the value of an existing attribute with the same name as the CoS attribute. The CoS definition entry takes the `cosSuperDefinition` object class and also inherits from the following object class that specifies the type of CoS:

cosClassicDefinition

The `cosClassicDefinition` object class determines the attribute and value that will appear with an entry by taking the base DN of the template entry from the `cosTemplateDN` attribute in the definition entry and combining it with the target entry specifier as defined with the `cosSpecifier` attribute, also in the definition entry. The value of the `cosSpecifier` attribute is another LDAP attribute which is found in the target entry; the value of the attribute found in the target entry is appended to the value of `cosTemplateDN` and the combination is the DN of the template entry. Template DNs for classic CoS must therefore have the following structure `cn=specifierValue,baseDN`.

CoS Template Entry

CoS Template entries are an instance of the `cosTemplate` object class. The CoS Template entry contains the value or values of the virtual attributes that will be generated by the CoS mechanism and displayed as an attribute of the target entry. The template entries are stored under the definition entries.

NOTE When possible, definition and template entries should be located at the same level for easier management.

Conflicts and CoS

There is the possibility that more than one CoS can be assigned to a role or organization, thus creating conflict. When this happens, Access Manager will display either the attribute value based on a pre-determined template priority level or the aggregate of all attribute values defined in the `cosPriority` attribute. For example, an administrator could create and load multiple services, register them to an organization, create separate roles within the organization and assign multiple roles to a particular user. When Access Manager retrieves this user entry, it sees the CoS object classes, and adds the virtual attributes. If there are any priority conflicts, it will look at the `cosPriority` attribute for a priority level and return the information with the lowest priority number (which is the highest priority level). For more information on CoS priorities, see "cosQualifier Attribute" on page 208 of Chapter 8, "Service Management" or the Sun Java System Directory Server documentation.

NOTE Conflict resolution is decided by the Directory Server before the entry is returned to Access Manager. Access Manager allows only the definition of the priority level and CoS type.

Glossary

For a list of terms that are used in this documentation set, refer to the *Sun Java™ Enterprise System Glossary* (`http://docs.sun.com/doc/816-6873`)

Index