Programmer's Guide

iPlanet™ Directory Server
Access Management Edition

Version 5.1

May 2002

Contents

About This Guide

This *Programmer's Guide* offers information on how to customize the iPlanet™ Directory Server Access Management Edition (DSAME) to fit the needs of each organization. This preface contains the following sections:

- What You Are Expected to Know
- iPlanet Directory Server Access Management Edition Documentation Set
- Organization of This Guide
- Documentation Conventions Used in This Guide
- Related Information

NOTE Sun™ One Identity Server was previously known as iPlanet Directory Server Access Management Edition (DSAME). The product was renamed shortly before the launch of the product.

The late renaming of this product has resulted in a situation where the new product name is not fully integrated into the shipping product. In particular, you will see the product referenced as DSAME within the product GUI and within the product documentation. For this release, please consider Sun One Identity Server and iPlanet Directory Server Access Management Edition as interchangeable names for the same product.

What You Are Expected to Know

This book is considered the "third" manual in the documentation series provided with iPlanet Directory Server Access Management Edition. It is intended for use by IT administrators and custom software developers who manage access to their web resources through iPlanet servers and software. The functionality allows the management of user data and enforcement of access policies. It is recommended that IS administrators understand directory server technologies, including

Lightweight Directory Access Protocol (LDAP), and have some experience with Java and eXtensible Markup Language (XML). Particularly, they should be familiar with Sun™ One Directory Server (DS) and the documentation provided with that product.

iPlanet Directory Server Access Management Edition Documentation Set

The Directory Server Access Management Edition documentation set contains the following titles:

- *Installation and Configuration Guide* describes iPlanet DSAME and provides details on how to plan and install the iPlanet DSAME on Solaris systems.
- *Administration Guide* documents how to manage user and service data and customize the DSAME console.
- *Programmer's Guide* (this guide) documents how to customize an iPlanet Directory Server Access Management Edition system for your organization.
- The *Release Notes* file gathers an assortment of information, including a description of what is new in this release, last minute installation notes, known problems and limitations, and how to report problems.

NOTE	Be sure to check the Directory Server Access Management Edition documentation web site for updates to the release notes and for revisions to the guides. Updated documents will be marked with the revision date. `http://docs.iplanet.com /docs/manuals/dsame.html`

Organization of This Guide

The *Programmer's Guide* (this guide) has nine chapters. The table below lists and briefly describes the content of these chapters.

Table 1 *Programmer's Guide* Chapters

Chapter	Description
About This Guide	An outline of the documentation set and a description of the *iPlanet Directory Server Access Management Edition Programmer's Guide*.
Chapter 1, "Introduction"	A brief explanation of the application's concepts.
Chapter 2, "DSAME And XML"	A description of how XML is used to customize the application.
Chapter 3, "User Authentication With DSAME"	A description of the Authentication module and how to create a custom authentication service.
Chapter 4, "Identity Management And The SDK"	A description of the management of identity-related objects and the DSAME SDK.
Chapter 5, "Single Sign-On With DSAME"	A description of the single sign-on and cross-domain single sign-on function and its APIs.
Chapter 6, "Logging"	A description of the Logging function and its APIs.
Chapter 7, "Utility APIs"	A description of the application's utility functions and its APIs.
Chapter 8, "iPlanet Directory Server And DSAME"	A description of the iPlanet Directory Server concepts that are used in DSAME.
Index	Alphabetical index of the *iPlanet Directory Server Access Management Edition Programmer's Guide*.

Documentation Conventions Used in This Guide

In the iPlanet Directory Server Access Management Edition documentation, there are certain typographic and terminology conventions used to simplify discussion and to help you better understand the material. These conventions are described below.

Typographic Conventions

This book uses the following typographic conventions:

- *Italic type* is used within text for book titles, new terminology, emphasis, and words used in the literal sense.

- `Monospace font` is used for sample code and code listings, API and language elements (such as function names and class names), filenames, pathnames, directory names, HTML tags, and any text that must be typed on the screen.
- *Italic serif font* is used within code and code fragments to indicate variable placeholders. For example, the following command uses *filename* as a variable placeholder for an argument to the gunzip command:

  gunzip -d *filename*.tar.gz

Terminology

Below is a list of the general terms that are used in the iPlanet Directory Server Access Management Edition documentation set:

- *DSAME* refers to iPlanet Directory Server Access Management Edition and any installed instances of the iPlanet Directory Server Access Management Edition software.
- *Policy and Management Services* refers to the collective set of iPlanet Directory Server Access Management Edition components and software you have installed and running on a dedicated Web Server.
- *Web Server that runs DSAME* refers to the dedicated Web Server where the DSAME is installed.
- *Directory Server* refers to an installed instance of iPlanet Directory Server or Netscape™ Directory Server.
- *DSAME_root* is a variable placeholder for the home directory where you have installed iPlanet Directory Server Access Management Edition.
- *Directory_Server_root* is a variable placeholder for the home directory where you have installed iPlanet Directory Server.
- *Web_Server_root* is a variable placeholder for the home directory where you have installed iPlanet Web Server.

Related Information

In addition to the documentation provided with iPlanet Directory Server Access Management Edition, you should be familiar with several other sets of documentation. Of particular interest are the iPlanet Directory Server, iPlanet Web Server, iPlanet Proxy Server, and iPlanet Certificate Management System documentation sets. This sections lists additional sources of information that can be used with iPlanet Directory Server Access Management Edition.

iPlanet Directory Server Documentation

You can find the iPlanet Directory Server documentation at the following site:

`http://docs.iplanet.com/docs/manuals/directory.html`

iPlanet Web Server Documentation

You can find the iPlanet Web Server documentation at the following site:

`http://docs.iplanet.com/docs/manuals/enterprise.html`

iPlanet Certificate Management System Documentation

You can find the iPlanet Certificate Management System documentation at the following site:

`http://docs.iplanet.com/docs/manuals/cms.html`

iPlanet Proxy Server Documentation

You can find the iPlanet Proxy Server documentation at the following site:

`http://docs.iplanet.com/docs/manuals/proxy.html`

Directory Server Developer Information

In addition to the Directory Server documentation, you can find information on Directory Server Access Management Edition, LDAP, the iPlanet Directory Server, and associated technologies at the following iPlanet developer sites:

`http://developer.iplanet.com/tech/directory/`

`http://www.iplanet.com/downloads/developer/`

Other iPlanet Product Documentation

Documentation for all iPlanet and Netscape servers and technologies can be found at the following web site:

`http://docs.iplanet.com/docs/manuals/`

iPlanet Technical Support

You can contact iPlanet Technical Support through the following location:

`http://www.iplanet.com/support/`

Chapter 1

Introduction

The *iPlanet Directory Server Access Management Edition* (DSAME) *Programmer's Guide* describes how service developers and programmers can customize the DSAME application to fit the specific needs of their organization. It offers information on the public Java APIs, XML-based service configuration files and HTML-based graphical interfaces. This introductory chapter contains the following sections:

- DSAME Overview
- Web Access
- Extending DSAME
- DSAME File System

DSAME Overview

iPlanet DSAME is designed to help organizations manage secure access to their web-based resources. The product integrates an identity system with the management and enforcement of authentication and access privileges. It contains a number of features towards this end. They include:

- Authentication—provides Java APIs for writing custom authentication server plug-ins, an HTML-defined client interface for gathering the user's credentials and a framework that connects the client interface with the plug-in module.
- Single Sign-on (SSO)—provides Java APIs to create and manage SSO tokens and a service to manage SSO sessions.

- Service Management—provides a solution for customizing and registering services and managing their *attributes* (configuration parameters). It includes an eXtensible Markup Language (XML) Document Type Definition (DTD) that defines the rules for creating a service and its attributes as well as Java APIs to manage the same.
- Identity Management—provides a solution for managing the structure of DSAME's directory data store. This includes Java APIs for adding, modifying and removing identity-related objects and their attributes as well as templates that define the configuration parameters of same.
- Policy Management—provides a solution for defining and retrieving organization-based privileges that secure the web resources of an enterprise.
- DSAME console—provides a graphical, JATO-based interface for identity, service, and policy management.
- Command-line interface—provides an `amadmin` executable to perform service schema and metadata integration as well as identity, policy and service management.

These listed features are executed by DSAME services that are installed out of the box. They would *internal* services as distinguished from customized or *external* services that are added on after installation. The basic functionality of *internal* services can be extended. Customized *external* components can be defined using the APIs, the sample code packaged with DSAME and the information in this guide.

How DSAME Works

DSAME can be used to manage access to resources in two ways. An administrator can access DSAME via a web browser or, an application can access DSAME directly, requesting user profile information.

Web Access

When a user requests access to an application or a protected page via a web browser, they must first be authenticated. The request is redirected to the Authentication service. This module determines the type of authentication to initiate based on the method chosen by the user's organization. For instance, LDAP is a simple user name and password-based authentication. The authentication module would send a HTML form to the web browser. For more complex types of authentication, it might send multiple forms for authentication information.

Having obtained the user's credentials, the Authentication module would call the respective provider to perform the authentication. Once verified, the module generates a Single Sign-On (SSO) token (using the SSO API) which holds the user's identity. The SSO API then generates a SSO token ID, a random string associated with the SSO token. This ID is then sent back to the browser in the form of a cookie. Once authenticated, the authentication component re-directs the user back to the requested application or page.

NOTE Web access through DSAME includes an additional security measure which uses web agents to evaluate a user's access privileges. For more information, see the iPlanet Policy Agent Pack 1.0 documentation.

Java Application Access

Java applications can access DSAME for user attributes. (For example, a mail service might store its users' mailbox size information in iPlanet Directory Server and retrieve the information using DSAME.) To achieve this, the system that runs the Java application must have the DSAME SDK installed. As well, there must be at least one instance of iPlanet Web Server running the DSAME internal services (specifically for the user authentication and SSO components).

Extending DSAME

DSAME can be extended in several ways. If additional authentication capabilities are needed, the Authentication APIs can be used to create them. To add Java-based applications, the SSO and Log APIs can be used to integrate them into the framework. The architectural goal of DSAME is to provide this extensible interface. This interface can be defined in one of three ways:

1. DSAME services are defined using XML.
2. DSAME screen templates are written using HTML.
3. DSAME services are implemented using Java.

Service Definition With XML

A DSAME service is a grouping of attributes defined under a common name. The attributes (or *configuration parameters*) can be a random set grouped together for easy management or a related set grouped together for a specific purpose. DSAME ships with a number of internal services of the latter type. These include, but are not limited to, logging, administration, and session services. More information on the internal services can be found in the *iPlanet Directory Server Access Management Edition Administration Guide*.

All DSAME services are written using the XML. The XML configuration file of a service must adhere to the form put forth in the `sms.dtd`, which is located in the *`Install_Directory/`*`SUNWam/dtd/` directory. Using the XML, organizations can modify the XML configuration files of internal DSAME services or configure the XML configuration files of external ones.

NOTE DSAME services manage attribute values stored in iPlanet Directory Server. They do not implement the behavior of the attributes or dynamically generate code to interpret them. It is up to an external application to interpret or utilize these values.

HTML Templates

DSAME uses HTML template files to control the look of the screens that a DSAME user sees. These templates can be modified to make changes to the design; for instance, an organization's logo can be added in place of the iPlanet logo. The entire template can also be replaced with an organization's custom HTML page.

The Java APIs

There are five public API packages provided with DSAME version 5.1. These APIs provide interfaces to implement the behavior of extended or customized DSAME services. The packages are introduced below.

Authentication API

DSAME allows the use of multiple and disparate authentication modules including, but not limited to, RADIUS, LDAP, Certificates, Unix, Membership (self-registration), SafeWord and Anonymous. Using the Authentication API, a service developer can write a custom authentication module. The API package name is `com.iplanet.authentication.spi`.

DSAME SDK

DSAME provides the framework to create and manage users, roles, groups, people containers, organizations, organization units, and sub-organizations. It also includes the functionality to create and modify service templates. This API is the core of the identity, service and policy management modules and provides Java classes that can be used to customize them. The API package name is `com.iplanet.am.sdk`.

Utilities API

This API provides a number of Java classes that can be used to manage system resources. This includes, among others, thread management and debug data formatting. The API package name is `com.iplanet.am.util`.

Logging API

The DSAME logging service records, among other things, access approvals, access denials and user activity. The Logging API can be used to enable other Java applications to call the DSAME logging service. The API package name is `com.iplanet.log`.

Single Sign-On API

DSAME provides Java interfaces for validating and managing the single sign-on (SSO) tokens and for maintaining the user's authentication credentials. All applications wishing to participate in the SSO solution can use this API. The API package name is `com.iplanet.sso`.

NOTE The Overview page for the complete set of public DSAME Javadocs can be accessed at *`Install_Directory`*`/SUNWam/docs/index.html`.

DSAME File System

DSAME installs its packages and files in a directory named `SUNWam`. The file system layout for a Solaris installation is as follows:

`Install_Directory``/SUNWam/`

- `bin/` ---> contains executables such as `amserver` & `amadmin`.
- `docs/` ---> contains DSAME documentation.
- `java/` ---> contains the Java Development Kit.
- `locale/` ---> contains the internationalization resource files.
- `servers/` ---> contains the iPlanet Web Server.
- `config/` ---> contains configuration files such as the iPlanet Directory Server name and port as well as XML files which define DSAME services.
- `dtd/` ---> contains the XML DTDs used by DSAME applications and services.
- `lib/` ---> contains DSAME jar files as well as platform specific C libraries.
- `migration/` ---> contains tools for iPlanet Directory Server data migration from earlier versions to version 5.1.
- `samples/` ---> contains sample java programs on how to use the DSAME Java APIs.
- `web-apps/` ---> contains two WAR-based deployments and their associated files: *Services* (authentication, policy management, identity management, SSO, SMS management, etc.) and *Applications* (DSAME console).

Runtime Directory

On Solaris, DSAME uses *`Install_Directory/`*`SUNWam` as its runtime directory for logs and debug files. On Windows 2000, DSAME uses *DSAME_root* as its runtime directory. Both directories can be configured.

DSAME performs three types of administration:

- Identity management deals with managing the structure of a customer's directory. This includes creating, deleting, and modifying roles, organizations.

- Policy management deals with the creation of policies and how they are applied within the application.
- Service management deals with service registration, unregistration, and activation.

Chapter 2

DSAME And XML

iPlanet Directory Server Access Management Edition (DSAME) uses Extensible Markup Language (XML) files for the integration and management of services. This chapter provides information on the XML service files installed with DSAME and the Document Type Definition (DTD) files used for creating new XML service files for the management of custom services. It contains the following sections:

- Overview
- Service Definition and Integration
- DSAME DTD Files
- DSAME XML Files
- Abstract Objects and amEntrySpecific.xml
- The amAdmin Command Line Executable
- SampleMailService Files

Overview

A *service* is a group of attributes, defined in an XML file, that are managed together by the DSAME console. The attributes can be the *configuration parameters* of a software module or they might just be related information with no relation to a software configuration. As an example of the first scenario, after creating a payroll module, a developer defines an XML service file that might include attributes to define an employee name, an hourly pay rate and a tax percentage. This file is imported into the iPlanet Directory Server (DS) so the attributes and their values can be stored. When the service is registered to an organization in DSAME, the attributes can be managed using the DSAME console.

NOTE Throughout this chapter, the term *attribute* is used as a modifier for two different concepts. A DSAME 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 the XML service files.

XML Service Files

XML service files enable DSAME to manage attributes that are stored in DS. DSAME does not implement any behavior or dynamically generate any code to interpret the attributes; it can only set or get attribute values. Out-of-the-box, DSAME loads a number of services to manage the attributes of its own internal modules. This includes, but is not limited to, the Logging, Authentication and User services. In addition to managing these attributes, DSAME provides code implementations to use them. For example, the URL Policy Access attributes are displayed and managed in the DSAME console, but the web agent itself is the code implementation using them to check user access to URLs. All DSAME-internal XML service files are located in `*Install_Directory*/SUNWam/config/xml`. For more specific information on the internal XML service files, see “Internal XML Service Files,” on page 58.

NOTE Any application with LDAP attributes can have data managed using the DSAME console by configuring a custom XML service file and loading it into the DS. For more information, see “Defining A Service,” on page 27.

Document Type Definition 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. DSAME exposes two DTD files which are used to define the structure for different types of XML files: `sms.dtd` and `amadmin.dtd`. The `sms.dtd` defines the structure for XML service files and the `amAdmin.dtd` defines the structure for XML files that are used to perform batch LDAP operations on the directory information tree (DIT) using the command line executable `amAdmin`. The DTDs are located in `*Install_Directory*/SUNWam/dtd`.

NOTE Knowledge of XML is necessary to understand the DTD elements and how they are integrated into DSAME. When creating an XML file, it might be helpful to print out the relevant DTD and a sample XML file made from the DTD.

Service Definition and Integration

Service Management in DSAME provides a mechanism for administrators to define, integrate and manage groups of attributes as a DSAME service. Readying a service for management involves creating an XML service file, configuring an LDIF file with any new object classes and importing both, the XML service file and the new LDIF schema, into the DS. Administrators can then register multiple services to organizations or sub-organizations using the DSAME console. Once registered, the attributes can be managed and customized per organization.

NOTE The only reason to create an XML service file is to group attributes to be managed using DSAME. If, for example, a software module has no attributes that need to be configured, no file is needed.

The following sections contain general information on DSAME service attributes as well as steps on how to define a service from configuration to registration.

Service Attributes

Services have different types of attributes. The `sms.dtd` structure enforces a service developer to define attributes as one of five types. The following sections provide descriptions of the five attribute types.

Global Attributes

Global attributes are defined for the DSAME 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 the DSAME 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 DSAME performs logging functions, the log files are written into a directory. The location of this directory is defined as a globally in the Logging service and all DSAME logs, independent of their purpose, are written to it. DSAME administrators can modify these default values through the Service Management page in the DSAME console. Global attributes are stored as an XML *blob* within an attribute of an LDAP object. Therefore, they do not need to be defined with a DS LDAP schema.

NOTE If a service has only global attributes, it can not be registered to an organization nor can a service template be created.

Organization Attributes

Organization attributes are defined and assigned at the organization level. Attributes for an Authentication service are a good example. When an Authentication service is registered, the 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 would be different depending on the address of an organization's LDAP server and the structure of their DIT, respectively. Organization attributes are stored as an XML *blob* within an attribute of an LDAP object. Therefore, they do not need to be defined with a DS LDAP schema.

NOTE Organization attributes are not inherited by sub-organizations. Only dynamic and policy attributes can be inherited. For additional information, see "Attribute Inheritance," on page 26.

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; roles have access to any service assigned to its parent organization. The dynamic attributes are then inherited by users that possess the role or belong to the organization. Because the attributes are assigned to roles or organizations instead of set in a user entry, they are *virtual* attributes inherited by users using *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*, both features of the iPlanet Directory Server. For information on these features, see Chapter 8, "iPlanet Directory Server And DSAME" or refer to the iPlanet 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 DS as LDAP objects, making it feasible to use traditional LDAP tools to manage them. A DS LDAP schema needs to be defined for these attributes.

Policy Attributes

Policy attributes are a special type of dynamic attribute. The main difference is that policy attributes provide a way to control resource access by defining a user's permissions. These defined permission attributes are then used to create *named policy*. For example, *allowURLList* is a named policy that defines a list of URLs a user is allowed to access; **.red.iplanet.com, *.eng.sun.com* are the permitted URLs defined as policy attributes. Named policies are assigned to roles or organizations; once assigned, the policy attribute is available in the user entry as an LDAP attribute, making it feasible to use traditional LDAP tools to manage them. (Named policies are not stored within the DS as LDAP objects.) A DS LDAP schema needs to be defined for these attributes.

Currently, DSAME has only two services that use policy attributes: URL Policy Agent and URL Domain Access. (Additionally, there is a sample mail service that uses policy attributes. For information on this sample, see "SampleMailService Files," on page 71.)

CAUTION Do not use the `policy.dtd` to define policy schema for a service. It is used internally for Policy Management.

User Attributes

User attributes belong specifically to 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. Examples of 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. User attributes can be a part of any service although DSAME has grouped a number of them into their own service defined by the `amUser.xml` service file. User attributes are stored within the DS as LDAP objects, making it feasible to use traditional LDAP tools to manage them. A DS 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 explicitly assigned to the user in order to display the attributes on the User's Profile page. In addition, the User Profile Display Option (defined in the Administration service) must be set to `Combined`. For more information, see the *iPlanet Directory Server Access Management Edition Administration Guide*.

Default Values

When a developer is writing an XML service file, default values can be defined for each attribute. After an XML service file is loaded into the DS, the default values can be displayed in the Service Management console. An organization can then register the service and create a service template where the default values can be modified. For example, all templates for the LDAP Authentication service use the `port` attribute. A default value of `389` could be defined in the XML service file and displayed on the LDAP Authentication Service Management page. Once registered to an organization, this value can be modified for the organization using the DSAME console. Default values are also used by integrated applications when a service template has not been created for an organization's registered service. For more information, see the "ChoiceValues Sub-Element" and the "DefaultValues Sub-Element," on page 40.

Attribute Inheritance

After creating and loading an XML service file, an administrator can assign the service's organization, dynamic and policy attributes by registering the service to an organization and creating a service template. The service, once registered, can be assigned to sub-organizations or a role. (Any number of services can be assigned to these objects.) When a user possesses a role or belongs to an organization which possesses a service, the user inherits the dynamic and policy attributes or the organization, dynamic and policy attributes, respectively. Inheritance only occurs, though, if the service possessed is also explicitly assigned to the user. A user can inherit attributes from multiple roles or parent organizations.

NOTE Attributes defined as *User* have no inheritance; they are set and modified in each User entry. For example, if 70 attributes are defined as *User* and an organization has two million users, each attribute is stored two million times.

ContainerDefaultTemplateRole

Dynamic and policy attributes are used in an XML service file if an administrator wants to define a service in which all DSAME user objects, with the specified service assigned to them, would inherit those attributes. After uploading the XML service file and assigning the service to an organization or role, all users in the sub-trees, with the specified service assigned to them, will inherit the dynamic and policy attributes. To accomplish this, DSAME uses classic CoS (as described in Chapter 8, “iPlanet Directory Server And DSAME) and role templates. `ContainerDefaultTemplateRole` is a default *filtered* role configured for each organization. The filter is `objectClass=iplanet-am-managed-person`. Since every user object in DSAME carries this attribute, every person in the organization possesses this role. DSAME then creates a separate CoS template for each registered service which points to the service’s default attributes. Any user who has the role will then get all the dynamic and policy attributes.

Defining A Service

The following procedures must be completed in order to use the DSAME console to integrate and manage a new service. They include creating or modifying XML files and registering these files using the `amadmin` command line executable.

1. Create an XML service file for the component.

   This XML 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. The file syntax can be found in “The sms.dtd Structure,” on page 34.

2. Extend the LDAP schema in the DS using `ldapmodify`, if necessary.

   Loading an LDIF file into the DS will add any new or modified object classes and attributes to the DIT. This step is only necessary when defining dynamic, policy and user attributes. Instructions on extending the LDAP schema can be found in “Extending The Directory Server Schema,” on page 28. See the Directory Server documentation for additional information.

3. Import the XML service file into DS using `amadmin` using the `--schema` or `-S` option.

   Information on importing an XML service file can be found in “Importing the XML Service File,” on page 31.

4. Configure a localization properties file and copy it into the `Install_Directory/SUNWam/locale` directory.

   The localization properties file must be created with accurate `i18nKey` fields that map to names defined in the XML service file. If no localization properties file exists, DSAME will display the actual attribute names. More information on the localization properties file can be found in "Configuring Localization Properties," on page 32.

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 DSAME *abstract objects*. Information on updating `amEntrySpecific.xml` can be found in "Abstract Objects and amEntrySpecific.xml," on page 65. 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 the latter case, `amUser.xml` would not need to be modified. Information on modifying `amUser.xml` can be found in "Modifying An Internal XML Service File," on page 60.

6. Register the service.

   After importing the service into DS, it can be registered and the attributes managed through the DSAME console. Information on how this can be done is in the *iPlanet Directory Server Access Management Edition Administration Guide*. Information on how to register using the command line can be found in "Registering the Service," on page 33.

Extending The Directory Server Schema

When configuring an XML service file for DSAME, it might also be necessary to extend the DS schema. Any dynamic, policy or user attributes defined in a DSAME service that are not already in the DS schema need to be added as LDAP object classes in order to store the data. This is done using the `ldapmodify` command line executable and an LDIF file as input.

NOTE	The order in which the LDAP schema is extended or the XML service file is loaded into DS is not important. Just remember that when a new service is loaded into DS, a complementary LDIF file should be created to load any new LDAP object classes.

1. Change to the DSAME `bin` directory.

   `cd` *`Install_Directory`*`/SUNWam/bin`

2. Create an LDIF file to define any new or modified LDAP object classes.

3. Run `ldapmodify` using the LDIF file as input.

   `ldapmodify -D` *`userid_of_DSmanager`* `-w` *`password`* `-f`
   `path_to_LDIF_file`

   By default, *`userid_of_DSmanager`* is `cn=Directory Manager`. If the schema was created correctly, the result of this command would be `Modifying entry cn=schema`.

NOTE After extending the schema using `ldapmodify`, it is not necessary to restart the DS. But, as `ldapmodify` is server-specific, the schema will not replicate and therefore needs to be extended on all configured servers. Additional information can be found in the iPlanet Directory Server documentation.

---

4. Run `ldapsearch` to ensure that the schema has been created.

   `ldapsearch -b “cn=schema” -s base -D` *`userid_of_DSmanager`* `-w`
   `password` `“(objectclass=*)” | grep -i “servicename”`

   If the schema was created correctly, the result of this command would be an LDIF listing of the object classes as displayed in Code Example 2-1 below.

Code Example 2-1 Sample Mail Service LDAP Object Class

```
objectClasses=( 1.2.3.888.23 NAME
'iplanet-am-sample-mail-service' DESC 'iPlanet SampleMail
Service' SUP topAUXILIARY MAY (
iplanet-am-sample-mail-service-status $
iplanet-am-sample-mail-root-folder
$iplanet-am-sample-mail-sentmessages-folder $
iplanet-am-sample-mail-indent-prefix
$iplanet-am-sample-mail-initial-headers $
iplanet-am-sample-mail-inactivity-interval $
iplanet-am-sample-mail-auto-load
$iplanet-am-sample-mail-headers-perpage $
iplanet-am-sample-mail-quota $
iplanet-am-sample-mail-max-attach-len
$iplanet-am-sample-mail-can-save-address-book-on-server )
X-ORIGIN ( 'iPlanet Directory Pro' 'user defined' ) )
attributeTypes=( 11.24.1.996.1 NAME
'iplanet-am-sample-mail-service-status' DESC 'iPlanet
SampleMailService Attribute'SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
X-ORIGIN ( 'iPlanet Directory Pro' 'user defined' ) )
```

Adding Object Classes To Existing Users

If a new service is created and the service's users already exist, the object classes need to be added to the user entries. In order to do this, DSAME provides migration scripts for performing batch updates to user entries in the DIT. No LDIF file need be created when using them. These scripts are described in the *iPlanet Directory Server Access Management Edition Installation and Configuration Guide.* Additionally, registered services can be added to each user by selecting the service from the specific user's Properties page.

To modify user entries using `ldapmodify`, an LDIF file would need to be created. Code Example 2-2 on page 31 shows how a user entry would be formatted in the LDIF file. This entry is having the object class `iplanet-am-sample-mail-service` and its attributes added.

Code Example 2-2 Sample LDIF File To Modify User Object

```
...
dn:cn=shiva,ou=People,o=iplanet.com,o=isp
       changetype:modify
       add:objectclass
       objectclass:iplanet-am-sample-mail-service
...
```

After creating the LDIF file, run `ldapmodify` as shown:

`Install_Directory``/SUNWam/bin/ldapmodify -D` *`userid_of_DSmanager`* `-w` *`password`* `-f` *`path_to_LDIF_file.`*

For more information on `ldapmodify` and user entries, see the iPlanet Directory Server documentation.

NOTE It is not recommended to use `ldapmodify` to create any entries for DSAME other than user entries.

Verifying The Directory Server Modification

To verify that the DIT has been populated correctly, an administrator can use `ldapsearch` or the following:

1. Change to the DS install directory:

   `cd /DS_`*`Install_Directory`*`/slapd-`*`DShostname`*

2. Export the DS contents into an LDIF file using:

   `db2ldif -s` *`orgnamingattribute=top_level_org_name`*

   This command results in the name of the LDIF file stored under *`DS_Install_Directory`*`/slapd-`*`DShostname`*`/ldif`. This file can be viewed to ensure that the objects described in the LDIF file have been created.

Importing the XML Service File

After creating an XML service file following the instructions in “DSAME DTD Files,” on page 33, the new service file needs to be imported into the DS.

1. Change to the DSAME install directory:

   `cd` *`Install_Directory`*`/SUNWam/bin`

2. Run:

```
./amadmin --runasdn DNofDSadministrator --password
passwordDSadministrator --verbose --schema xmlservicefilepath
```

NOTE If changing an existing service, the original XML service file needs to be deleted before importing the modified XML service file.

Configuring Localization Properties

A localization properties file specifies the locale-specific screen text and messages that an administrator or user will see when directed to a service's attribute configuration page. The files are located in the *Install_Directory*/`SUNWam/locale/` directory.

Code Example 2-3 Sample Mail Service Localization Properties File

```
...
iplanet-am-sample-mail-service-description=Sample Mail Service
Profile
a1=Mail Status
a2=Root Folder
a3=Sent Messages Folder
a4=Reply Prefix
a5=Initial Headers to Load
a6=Check New Mail Interval (minutes)
a7=Automatic Message Load at Disconnect
a8=Headers Per Page
p1=Mail Quota
p2=Auto-download Maximum Attachment Length
p3=Save Address Book on Server
```

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 DSAME console. The keys (`a1`, `a2`, etc.) map to the `i18nKey` attribute fields defined for a service. Code Example 2-3 is the localization properties file for DSAME's sample mail service. The keys also determine the order in which the fields are displayed, taken in alphabetical and then numerical order (`a1`, `a2` is followed by `b1`, `b2` and so forth). Note that the keys are strings, so `a10` comes before `a2`.

NOTE If modifying a localization properties file, DSAME needs to be restarted. If importing a new service, DSAME does not need to be restarted to recognize the localization properties file.

Identifying The Localization Properties File

DSAME also needs to be able to locate the localization properties file. An administrator needs to ensure that it is located in the default *`Install_Directory/`*`SUNWam/locale` directory. If the file is kept in another directory, the `jvm.classpath=` entry in the `jvm12.conf` file needs to be modified to include the new directory pathname.

NOTE	If the `jvm12.conf` file is modified, the DSAME server needs to be restarted.

Registering the Service

The preferred way to register a service is to use the DSAME console. Information on how this is done can be found in the *iPlanet Directory Server Access Management Edition Administration Guide*. Alternately, services can be registered using the `amadmin` command line executable.

1. Change to the DSAME install directory:

   cd *Install_Directory/*SUNWam/bin

2. Run:

   amadmin --runasdn *DNofDSAMEadministrator* --password *passwordDSAMEadministrator* --schema *path_to_xmlservicefile*

DSAME DTD Files

DSAME contains two DTD files which are used to define the structure for XML files used within the configuration. The `sms.dtd` defines the structure for XML service files and the `amAdmin.dtd` defines the structure for XML files that are used to perform batch LDAP operations on the directory information tree (DIT) using the command line executable `amAdmin`. The DTDs are located in *`Install_Directory/`*`SUNWam/dtd`.

CAUTION	Neither of these DTD files should be modified in any way. They contain rules and definitions that control how certain operations are performed on the DIT and any alterations might hinder them.

Code Examples. DSAME comes with the files needed to integrate a mail service into the configuration. These sample files are used throughout this section to illustrate the DTD concepts. For more information on these files, see "SampleMailService Files," on page 71.

The sms.dtd Structure

The `sms.dtd` defines the data structure for all XML service files. It is located in the *`Install_Directory/`*`SUNWam/dtd` directory. The `sms.dtd` enforces the developer to define each attributes as one of five schema types which are then stored and managed differently. For instance, some of the attributes are applicable to an entire DSAME 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
- User
- Policy

An explanation of the elements defined by the `sms.dtd` follows. Each element includes a number of XML attributes which are also explained. DSAME currently supports only about 20% 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 DSAME converts all attribute names to lower case when reading from the DS.

ServicesConfiguration Element

ServicesConfiguration is the root element of the XML service file. It's immediate sub-element is *Service*. Code Example 2-4 on page 35 is the *ServicesConfiguration* element as defined in the `sampleMailService.xml`.

Code Example 2-4 Element ServicesConfiguration and Element Service

```
...
<ServicesConfiguration>
<Service name=”sampleMailService” version=”1.0”>
<Schema...>
...
```

Service Element

The *Service* element defines the schema for a given service. Multiple services can be defined in a single XML file with this element, but it is recommended that only one be defined per XML service file. Currently, DSAME supports the sub-element *Schema* which, in turn, defines DSAME attributes as either *Global*, *Organization*, *Dynamic*, *User* or *Policy*. The required XML service attributes for the *Service* element are the name of the service, such as *iPlanetAMLoggingService*, and the version number of the XML service file itself. Code Example 2-4 on page 35 illustrates the *Service* element and its attributes as defined in the `sampleMailService.xml`.

Schema Element

The *Schema* element is the parent of the elements that define the service’s specific DSAME attributes (global, organization, dynamic, user or policy) and their default values. The sub-elements can be *Global*, *Organization*, *Dynamic*, *User* or *Policy*. The required XML attributes of the *Schema* element include `serviceHierarchy` which defines where the service will be displayed in the DSAME console, `i18nFileName` which defines the name of the localization properties file, and `i18nKey` which defines the attribute in the localization properties file from which this particular defined value will be taken.

NOTE The *Schema* element is required in XML service files.

serviceHierarchy Attribute

When adding a service, this attribute must be defined in order to display the service in the DSAME console. When a new service is registered, it is dynamically displayed based on this value. The value is a "/" separated string. Code Example 2-5 on page 36 illustrates the `serviceHierarchy` attribute as defined in the `sampleMailService.xml`. The name *sampleMailService* is used to find the localization properties file which defines what will be displayed below the Other Configuration header in the DSAME console.

Code Example 2-5 `i18nFileName`, `i18nKey` and `serviceHierarchy` Attributes

```
...
<Schema
 serviceHierarchy="/other.configuration/sampleMailService"
           i18nFileName="sampleMailService"
           i18nKey="iplanet-am-sample-mail-service-description">
...
```

i18nFileName And i18nKey Attributes

These two XML attributes both refer to the localization properties files. The `i18nFileName` attribute takes a value equal to the name of the localization properties file for the defined service (minus the `.properties` file extension). The `i18nKey` is a text string that maps to a property value defined in the localization properties file (specified, as discussed, in the `i18nFileName` attribute.) For example, Code Example 2-5 on page 36 defines the name of the properties file as `sampleMailservice` and the text-based value of the `i18nKey` maps to its final value as defined in `sampleMailservice.properties`. The final value is the name of the service as it will be displayed in the DSAME console; in this case, *Sample Mail Service Profile* is the name defined in `sampleMailservice.properties`. More information on the localization properties file can be found in "Configuring Localization Properties," on page 81 of Chapter 3, "User Authentication With DSAME."

Schema Sub-Elements

The next five elements are sub-elements of *Schema*; they are the declarations of the service's DSAME attributes. When defining a service, each attribute must be defined as one of these types: Global, Organization, Dynamic, Policy and User. Any configuration (all or none) of these elements can be used depending on the service. Each DSAME attribute defined within these elements is itself defined by the sub-element AttributeSchema or, in the case of Policy, the ActionSchema.

Global Element

The Global element defines DSAME 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 23.

serviceObjectClasses Attribute. The `serviceObjectClasses` attribute is a global attribute in each XML service file that contains dynamic or policy attributes. This optional attribute is used by the SDK to set the object class for the service in the user entries. When an organization registers a service with the `serviceObjectClasses` attribute defined, the service's dynamic or policy attributes, if any exist, are automatically assigned to any user object which has been assigned the service. If the `serviceObjectClasses` attribute is not specified and the service has defined dynamic or policy attributes, an object class violation is called when an administrator tries to create a user under that organization.

Multiple values can be defined for the *serviceObjectClasses* attribute. For example, if a service is created with two attributes each from three different object classes, the `serviceObjectClasses` attribute would need to list all three object classes as `DefaultValues`. Code Example 2-6 has two defined object classes.

Code Example 2-6 `serviceObjectClass` Defined As Global Element

```
...
<Global>
            <AttributeSchema name="serviceObjectClasses"
                type="list"
                syntax="string"
                i18nKey="">
                <DefaultValues>
                    <Value>iplanet-am-sample-mail-service</Value>
                   <Value>iplanet-am-other-sample-service</Value>
                </DefaultValues>
            </AttributeSchema>
 </Global>
 ...
```

Organization Element

The Organization element defines DSAME attributes that are modifiable per organization or sub-organization. For example, a web hosting environment using DSAME 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 DSAME SDK. For additional information, see "Organization Attributes," on page 24.

Dynamic Element

The Dynamic element defines DSAME 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 always use the DS features, CoS (Class Of Service) and Roles. For additional information, see "Dynamic Attributes," on page 24.

User Element

The User element defines DSAME 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 25.

Policy Element

The Policy element defines DSAME attributes intended to provide 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 include `canChangeSalaryInformation` and `canForwardEmailAddress`. See Code Example 2-8 on page 41 for an example of a Policy schema definition from the `sampleMailService.xml` file. For additional information, see "Policy Attributes," on page 25.

SubSchema Element

The `SubSchema` element can specify multiple subschemas of global information for different defined applications. For example, logging for a calendar application could be separated from logging for a mail service application. Another example would be a service developer defining choice values for different logging levels. For logging `type`, choice values can be defined to specify output that goes to a file, JDBC, or some other LDAP output mechanism. The attribute `multiple_choice` represents a list of choice values. The choice values could represent multiple values, so that if the attribute values do not contain multiple choice values, then the SMS parsing would fail.

NOTE The Service 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 “Schema Sub-Elements,” on page 36. It defines the structure of each attribute. The sub-elements that qualify the AttributeSchema can include `IsOptional?`, `IsServiceIdentifier?`, `IsStatusAttribute?`, `ChoiceValues?`, `BooleanValues?`, `DefaultValues?`, or `Condition`. The XML attributes that define each portion of the attribute value are `name`, `type`, `syntax`, `cosQualifier`, `rangeStart`, `rangeEnd`, `validator`, `any`, and `%i18nIndex`. Code Example 2-7 on page 39 illustrates the *AttributeSchema* element, its attributes and their corresponding values. Note that this example attribute is a Dynamic attribute.

Code Example 2-7 `AttributeSchema` Element With XML Attributes

```
...
<Dynamic>
  <AttributeSchema name="iplanet-am-sample-mail-service-status"
                type="single_choice"
                syntax="string"
                i18nKey="a1">
                <ChoiceValues>
                    <ChoiceValue>Active</ChoiceValue>
                    <ChoiceValue>Inactive</ChoiceValue>
                    <ChoiceValue>Deleted</ChoiceValue>
                </ChoiceValues>
                <DefaultValues>
                    <Value>Active</Value>
                </DefaultValues>
 </AttributeSchema>
</Dynamic>
...
```

name Attribute

This required XML attribute defines the LDAP name for the attribute. Any string format can be used but attribute names must be in lower-case. Code Example 2-7 on page 39 defines it with a value of `iplanet-am-sample-mail-service-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 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 chose a single value from a list of options.
- `multiple_choice` specifies that the user can chose multiple values from a list of options.

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. 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 ChoiceValue element. Code Example 2-7 on page 39 defines the attribute type as `single_choice` so the `ChoiceValues` attribute defines the list of options as `Active`, `Inactive` and `Deleted`.

syntax Attribute

The `syntax` attribute defines the format of the value. The default value for syntax is `string` but, it can be defined as 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 .
- `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.

DefaultValues Sub-Element. Defining any of these syntax values also necessitates defining a value for the DefaultValue sub-element. A default value will then be displayed in the DSAME console but can be changed for each organization when creating a new template for the service. In the Code Example 2-8 on page 41, for example, the *Save Address Book On Server* field will display a default value of `false`. The user has the option to change the value to `true`, if desired. (The default value for `password` would be an encrypted password, generally the same as the one used for DSAME.)

Code Example 2-8 `ActionSchema` Element With Boolean Syntax

```
...
<ActionSchema
name="iplanet-am-sample-mail-can-save-address-book-on-server"
                    type="single"
                    syntax="boolean"
                    i18nKey="p3">
                    <DefaultValues>
                        <Value>false</Value>
                    </DefaultValues>
                </ActionSchema>
...
```

The `ActionSchema` as displayed in Code Example 2-8 on page 41 is discussed in "ActionSchema Element," on page 43.

cosQualifier Attribute

This attribute defines how DSAME will resolve conflicting `cosQualifier` attributes assigned to the same user object. This value will appear as a qualifier to the `cosAttribute` in the LDAP entry of the CoS definition. It can be defined as:

- `default` indicates that if there are two conflicting `cosQualifier` attributes assigned to the same user object, the one with the lowest priority number (0) takes precedence. (The priority level is set in the `cosPriority` attribute when a new CoS template entry is created for an organization or role. For more information, see "Conflicts and CoS," on page 143 of Chapter 8, "iPlanet Directory Server And DSAME.")
- `override` indicates that the CoS template value overrides any value already present in the user entry; that is, CoS takes precedence over the 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 templates.

NOTE The URL Policy Agent service uses `merge-schemes` to obtain aggregated values for the Allow and Deny attributes. For example, if the Employee Role allows access to `*/employee.html` and the HR Role allows access to `*/hr.html`, a user possessing both of these roles is allowed access to both.

If this 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 DSAME console. It has six possible values that can be multiply defined using the "|" (pipe) construct:

- `display` specifies that the attribute will display on the user profile page. The attribute is read / write for administrators and regular users.
- `adminDisplay` specifies that the attribute will display on the user profile page. It will not appear on an 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 regular users. It is displayed on the user profile pages as a non-editable label for regular users.
- `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.
- `optional` specifies that a value for the attribute is not required in order for the object to be created.
- `filter` specifies that the attribute will display on the 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 DSAME 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 DSAME 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.

%i18nIndex Attribute (i18nKey)

The `i18nKey` attribute, as defined in "i18nFileName And i18nKey Attributes," on page 36, is referenced as an entity in the `sms.dtd`.

NOTE If the `i18nKey` value is blank (that is, " "), the DSAME console will not display the attribute.

ActionSchema Element

The *ActionSchema* element is a sub-element of the Policy attribute element discussed in "Policy Element," on page 38. It defines the structure of Policy attributes only. The sub-elements that qualify the ActionSchema can include `IsOptional?`, `ActionValue?`, `BooleanValues?`, and `DefaultValues?` The XML attributes that define each portion of the attribute value are `name`, `type`, `syntax`, `cosQualifier`, `rangeStart`, `rangeEnd`, `validator`, `any`, and `%i18nIndex`. Code Example 2-8 on page 41 illustrates the *ActionSchema* element, its attributes and their corresponding values.

NOTE The difference between *AttributeSchema* and *ActionSchema* is that the *ActionSchema* element has Policy-specific attributes, such as `ActionValue`, and the *AttributeSchema* has attributes not applicable to Policy, such as `IsStatusAttribute`?

ResourceName Element

The ResourceName element specifies if the service has resources associated with it, for example, URLs in the case of URL Policy Agent service.

The amAdmin.dtd Structure

The `amAdmin.dtd` defines the data structure for all XML files which will be used to perform batch LDAP operations on the DIT using `amAdmin`. It is located in the *`Install_Directory`*`/SUNWam/dtd` directory. The command line operations include reads and gets on the attributes as well as creations and deletions of user objects (roles, organizations, users, people containers, and groups). The following sections discuss the elements and attributes of the amAdmin.dtd as well as the sample XML templates installed with DSAME that use this structure. These samples can be found in *Install_Directory*/SUNWam/samples/admin/cli/bulk-ops and will be used to illustrate these sections.

Requests Element

The *Requests* element is the root element of the batch processing XML file. It must contain at least one child element which defines the DSAME identity objects (Organization, Container, People Container, Role and Group) onto which the actual requests are performed. To enable batch processing, the root element can take more than one set of requests. The *Requests* element must contain at least one of the following sub-elements:

- OrganizationRequests
- ContainerRequests
- PeopleContainerRequests
- RoleRequests
- GroupRequests
- SchemaRequests
- ServiceConfigurationRequests

Based on the defined request, the corresponding DSAME API will be called to perform the operation.

OrganizationRequests Element

The *OrganizationRequests* element consists of all requests that can be performed on Organization objects. The required XML attribute for this element is the LDAP Distinguished Name (DN) of the organization on which all of the sub-element requests will be performed. This element can have one or more sub-elements which perform their operations on the defined instance of the Organization object. (Different *OrganizationRequests* elements can be defined in one document to modify more than one Organization DN.) Code Example 2-9 on page 49 defines a myriad of objects to be created from the top level organization, o=isp. The sub-elements of *OrganizationRequests* are:

- CreateSubOrganization
- CreatePeopleContainer
- CreateRole

- `CreateGroup`
- `CreatePolicy`
- `AssignPolicy`
- `UnAssignPolicy`
- `CreateServiceTemplate`
- `ModifySubOrganization`
- `ModifyServiceTemplate`
- `DeleteServiceTemplate`
- `ModifyPeopleContainer`
- `ModifyRole`
- `ModifyGroup`
- `ModifyPolicy`
- `GetSubOrganizations`
- `GetPeopleContainers`
- `GetRoles`
- `GetGroups`
- `GetUsers`
- `RegisterServices`
- `UnregisterServices`
- `GetRegisteredServiceNames`
- `GetNumberOfServices`
- `DeleteRoles`
- `DeleteGroups`
- `DeletePolicy`
- `DeletePeopleContainers`
- `DeleteSubOrganizations`

ContainerRequests Element

The *ContainerRequests* element consists of all requests that can be performed on Container objects. The required XML attribute for this element is the `DN` of the container on which the sub-element requests will be performed. This element can have one or more sub-elements which perform their operations on the same instance of the container. (Different *ContainerRequests* elements can be defined in one document to modify more than one Container DN.) Code Example 2-9 on page 49 illustrates how this element can be modeled. The sub-elements of *ContainerRequests* are:

- `CreateSubContainer`
- `CreatePeopleContainer`
- `CreateRole`
- `CreateGroup`
- `CreatePolicy`
- `AssignPolicy`
- `UnAssignPolicy`
- `CreateServiceTemplate`
- `ModifyServiceTemplate`
- `ModifySubContainer`
- `ModifyPeopleContainer`
- `ModifyRole`
- `GetSubContainers`
- `GetPeopleContainers`
- `GetRoles`
- `GetGroups`
- `GetUsers`
- `RegisterServices`
- `UnregisterServices`
- `GetRegisteredServiceNames`
- `GetNumberOfServices`

- `DeleteRoles`
- `DeleteGroups`
- `DeletePolicy`
- `DeletePeopleContainers`
- `DeleteSubContainers`

PeopleContainerRequests Element

The *PeopleContainerRequests* element consists of all requests that can be performed on People Container objects. The required XML attribute for this element is the `DN` of the container on which the sub-element requests will be performed. This element can have one or more sub-elements which perform their operations on the same instance of the people container. (Different *PeopleContainerRequests* elements can be defined in one document to modify more than one People Container DN.) Code Example 2-9 on page 49 illustrates how this element can be modeled. The sub-elements of *PeopleContainerRequests* are:

- `CreateSubPeopleContainer`
- `ModifyPeopleContainer`
- `CreateUser`
- `ModifyUser`
- `GetNumberOfUsers`
- `GetUsers`
- `GetSubPeopleContainers`
- `DeleteUsers`
- `DeleteSubPeopleContainers`

RoleRequests Element

The *RoleRequests* element consists of all requests that can be performed on roles. The required XML attribute for this element is the `DN` of the role on which the sub-element requests will be performed. This element can have one or more sub-elements which perform their operations on the same instance of the role. (Different *RoleRequests* elements can be defined in one document to modify more than one Role DN.) Code Example 2-9 on page 49 illustrates how this element can be modeled. The sub-elements of *RoleRequests* are:

- `CreateServiceTemplate`

- `ModifyServiceTemplate`
- `AssignPolicy`
- `UnAssignPolicy`
- `GetNumberOfUsers`
- `GetUsers`
- `AddUsers`

GroupRequests Element

The *GroupRequests* element consists of all requests that can be performed on group objects. The required XML attribute for this element is the `DN` of the group on which the sub-element requests will be performed. This element can have one or more sub-elements which perform their operations on the same instance of the group. (Different *GroupRequests* elements can be defined in one document to modify more than one Group DN.) Code Example 2-9 on page 49 illustrates how this element can be modeled. The sub-elements of *GroupRequests* are:

- `CreateSubGroup`
- `GetSubGroups`
- `GetNumberOfUsers`
- `GetUsers`
- `AddUsers`
- `DeleteSubGroups`

AttributeValuePair Element

The *AttributeValuePair* element can be a sub-element of many of the following batch processing requests. It can have two sub-elements, neither of which can themselves have sub-elements. The *Attribute* sub-element must be empty while the *Value* sub-element takes a default value to display in the DSAME console. The *Attribute* sub-element takes a required XML attribute called `name`. The value of `name` is the attribute name which is equal to one string without spaces; no sub-elements are allowed. Code Example 2-14 on page 53 illustrates how an attribute/value pair would be added to a sub-organization.

Create*Object* Elements

The *CreateSubOrganization*, *CreateUser*, *CreateGroup*, *CreateSubContainer*, *CreatePeopleContainer*, *CreateSubGroup*, *CreateSubPeopleContainer* and *CreateRole* elements create a sub-organization, user, group, sub-container, people container, sub-group, sub-people container and role, respectively. The object is created in the DN that is defined in the second-level *<Object>Requests* element under which the *Create<Object>* element is defined. *AttributeValuePair* may be defined as a sub-element (or not). The required XML attribute for each element is `createDN`; it takes the `DN` of the object to be created. Code Example 2-9 on page 49 illustrates an example of some of these elements.

Code Example 2-9 Portion of Batch Processing File `createRequests.xml`

```
...
<Requests>
<OrganizationRequests DN="o=isp">

    <CreateSubOrganization createDN="sun.com"/>
    <CreatePeopleContainer createDN="People1,o=sun.com"/>
    <CreateRole createDN="ManagerRole,o=sun.com"/>
    <CreateRole createDN="EmployeeRole,o=sun.com"/>
    <CreateGroup createDN="ContractorsGroup,o=sun.com"/>
    <CreateGroup createDN="EmployeesGroup,o=sun.com"/>

</Requests>
```

CreatePolicy Element

The *CreatePolicy* element creates one or more policy attributes. The *Policy* sub-element defines the named policy. The required XML attribute is `createDN` which takes the `DN` of the organization where the policy will be created. This and the following nested elements are all illustrated in Code Example 2-10 on page 50.

Policy Element. The *Policy* sub-element defines the permissions or *rules* of the policy. It can take one or more of the *Rule* sub-elements. The required XML attribute is `name` which specifies the name of the policy. The `serviceName` attribute, which identifies the service to which the named policy applies, is an optional XML attribute.

Rule Element. The *Rule* sub-element defines a specific permission of the policy. *Rule* 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* sub-element defines the service for which a rule has been created. There are no sub-elements; the *ServiceName* element itself must be empty. The required XML attribute is `name` which takes a string value.

- ResourceName Element

  The *ResourceName* sub-element defines the domain for which this permission is being defined. There are no sub-elements; the *ResourceName* element itself must be empty. The required XML attribute is `name` which takes a string value.

- AttributeValuePair Element

  The *AttributeValuePair* sub-element defines the action names and corresponding action values of the rule. For additional information, see "DeleteObject Elements," on page 51.

Code Example 2-10 Portion of Batch Processing File `createPolicyOrg.xml`

```
...
<Requests>

<OrganizationRequests DN="o=isp">

<CreatePolicy createDN="o=iplanet.com,o=isp">
 <Policy name="urlpolicy" serviceName="iPlanetAMWebAgentService">
             <Rule name="Manager Rule">
                 <ServiceName name="iPlanetAMWebAgentService"/>
                 <ResourceName name="*.red.iplanet.com"/>
                 <AttributeValuePair>
                     <Attribute name="permission"/>
 <Value>iplanet-am-web-agent-access-allow-list</Value>
                 </AttributeValuePair>
             </Rule>
             <Rule name="engManager Rule">
                 <ServiceName name="iPlanetAMWebAgentService"/>
                 <ResourceName name="*.eng.iplanet.com"/>
                 <AttributeValuePair>
                     <Attribute name="permission"/>
 <Value>iplanet-am-web-agent-access-allow-list</Value>
                 </AttributeValuePair>
             </Rule>
        </Policy>
    </CreatePolicy>
</OrganizationRequests>
</Requests>
```

CreateServiceTemplate Element

The *CreateServiceTemplate* element creates a service template for the organization defined in 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 2-11 on page 51 illustrates a service template being created for `sun.com`.

Code Example 2-11 Portion of Batch Processing File `createServiceTemplates.xml`

```
...
<Requests>
<OrganizationRequests DN="o=sun.com,o=isp">

    <CreateServiceTemplate serviceName="sampleMailService"/>

</OrganizationRequests>
</Requests>
...
```

Delete*Object* Elements

The *DeleteSubOrganizations*, *DeleteUsers*, *DeleteGroups*, *DeleteSubContainers*, *DeletePeopleContainers*, *DeleteSubGroups*, *DeleteSubPeopleContainers*, and *DeleteRoles* elements delete a sub-organization, user, group, sub-container, people container, sub-group, sub-people container and role, respectively. The object is deleted from the DN that is defined in the second-level *<Object>Requests* element under which the *Delete<Object>* element is 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 subtrees 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 2-12 on page 51 illustrates an example of some of these elements.

Code Example 2-12 Portion of Batch Processing File deleteOrgRequest`s.xml`

```
...
<Requests>
<OrganizationRequests DN="o=isp">

    <DeleteRoles>
       <DN>cn=ManagerRole,o=sun.com,o=isp</DN>
       <DN>cn=EmployeeRole,o=sun.com,o=isp</DN>
    </DeleteRoles>
```

Code Example 2-12 Portion of Batch Processing File deleteOrgRequests.xml

```
...
<Requests>
    <DeleteGroups deleteRecursively="true">
       <DN>cn=EmployeesGroup,o=sun.com,o=isp</DN>
       <DN>cn=ContractorsGroup,o=sun.com,o=isp</DN>
    </DeleteGroups>

    <DeletePeopleContainers deleteRecursively="true">
       <DN>ou=People1,o=sun.com,o=isp</DN>
    </DeletePeopleContainers>

    <DeleteSubOrganizations deleteRecursively="true">
       <DN>o=sun.com,o=isp</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 2-13 on page 52 illustrates how this element is formatted.

Code Example 2-13 Portion of Batch Processing File `deleteServiceTemplates.xml`

```
...
<Requests>
<OrganizationRequests DN="o=iplanet.com,o=isp">
      <DeleteServiceTemplate
serviceName="iPlanetAMAuthLDAPService"
                schemaType="organization">

      </DeleteServiceTemplate>
</OrganizationRequests>
</Requests>
```

Modify*Object* Elements

The *ModifyPeopleContainer*, *ModifySubContainer*, *ModifySubOrganization* and *ModifyRole*, *ModifyGroup* elements change the specified object. *AttributeValuePair* can be defined as a sub-element of the first four listed elements. (The *ModifyGroup* element can have no sub-elements; it must be empty.) The required XML attribute is `modifyDN` which takes the `DN` of the object to be modified. Code Example 2-14 on page 53 illustrates how these elements can be modeled.

Code Example 2-14 Portion of Batch Processing File `modifyRequests1.xml`

```
...
<Requests>
<OrganizationRequests DN="o=isp">

    <ModifySubOrganization modifyDN="o=sun.com,o=isp">
      <AttributeValuePair>
         <Attribute name="Description"/>
         <Value>DSAME Modify</Value>
      </AttributeValuePair>
    </ModifySubOrganization>

    <ModifyPeopleContainer modifyDN="People1,o=sun.com">
      <AttributeValuePair>
         <Attribute name="Description"/>
         <Value>DSAME Modify</Value>
      </AttributeValuePair>
    </ModifyPeopleContainer>

    <ModifyRole modifyDN="ManagerRole,o=sun.com">
      <AttributeValuePair>
         <Attribute name="iplanet-am-role-description"/>
         <Value>DSAME Modify</Value>
      </AttributeValuePair>
    </ModifyRole>


</OrganizationRequests>
</Requests>
```

ModifyServiceTemplate Element

The *ModifyServiceTemplate* element changes a specified service template. *AttributeValuePair* must be defined as a sub-element of *ModifyServiceTemplate* to change the values. The required XML attribute is `serviceName` which takes a string value and `schemaType`. Code Example 2-15 on page 54 illustrates this element.

Code Example 2-15 Portion of Batch Processing File `modifyServiceTemplates.xml`

```
...
<Requests>
<OrganizationRequests DN="o=sun.com,o=isp">

   <ModifyServiceTemplate serviceName="sampleMailService">
     <AttributeValuePair>
       <Attribute
name="iplanet-am-sample-mail-sentmessages-folder"/>
       <Value>Hello Mail Sent</Value>
    </AttributeValuePair>
   </ModifyServiceTemplate>
</OrganizationRequests>
</Requests>
```

Get*Object* Elements

The *GetSubOrganizations, GetUsers, GetGroups, GetSubContainers, GetPeopleContainers* and *GetRoles* 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 within the organization defined in the second-level *Requests* element will be returned. The required XML attribute for all but *GetGroups* and *GetRoles* is `DNsOnly` and takes a `true` or `false` value. The required XML attribute of *GetGroups* and *GetRoles* is `level` which takes a value of either `ONE_LEVEL` or `SUB_TREE`. `ONE_LEVEL` will retrieve just the groups at that node level; `SUB-TREE` gets groups at the node level and all those underneath it. Code Example 2-16 on page 55 illustrates how these elements can be modeled.

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 2-16 Portion of Batch Processing File `getRequests.xml`

```
...
<Requests>

<OrganizationRequests DN="o=isp">

    <GetSubOrganizations DNsOnly="false">
         <DN>o=iplanet.com,o=isp</DN>
         <DN>o=sun.com,o=isp</DN>
    </GetSubOrganizations>

    <GetPeopleContainers DNsOnly="false">
         <DN>ou=People,o=iplanet.com,o=isp</DN>
         <DN>ou=People,o=sun.com,o=isp</DN>
    </GetPeopleContainers>

    <GetRoles level="SUB_TREE"/>

    <GetGroups level="SUB_TREE"/>

    <GetUsers DNsOnly="false">
        <DN>cn=puser,ou=People,o=iplanet.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 2-17 on page 55 illustrates how the *GetRegisteredServiceNames* element is modeled.

Code Example 2-17 Batch Processing File `getRegisteredServiceNames.xml`

```
...
<Requests>

<OrganizationRequests DN="o=sun.com,o=isp">
    <GetRegisteredServiceNames/>
</OrganizationRequests>

</Requests>
```

*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 2-18 on page 56 illustrates how these elements can be modeled.

Code Example 2-18 Portion of Batch Processing File `registerRequests.xml`

```
...
<Requests>
<OrganizationRequests DN="o=sun.com,o=isp">

    <RegisterServices>
        <Service_Name>sampleMailService</Service_Name>
    </RegisterServices>

</OrganizationRequests>
</Requests>
```

AssignPolicy and UnAssignPolicy Elements

The *AssignPolicy* and *UnAssignPolicy* elements take 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 required XML attribute of *AssignPolicy* and *UnAssignPolicy* is `policyDN` which takes a value equal to the DN of the policy to be acted upon.

SchemaRequests Element

The *SchemaRequests* element consists of all requests that can be performed on the default values of the DSAME schema. It has two required XML attributes: *serviceName* and *SchemaType*. *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). This element can have zero or more sub-elements. The sub-elements of *SchemaRequests* are:

- `RemoveDefaultValues Element`
- `ModifyDefaultValues Element`
- `AddDefaultValues Element`
- `GetServiceDefaultValues`

RemoveDefaultValues Element

The *RemoveDefaultValues* element removes the default values from the schema specified in the parent *SchemaRequests* element. It takes a sub-element of *Attribute* which specifies the name of the attribute to be removed. The *Attribute* sub-element itself must be empty; it takes no sub-element. There is no required XML attribute.

Code Example 2-19 Portion of Batch Processing File `removeschemaRequests.xml`

```
...
<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 a sub-element of *AttributeValuePair* which specifies the name of the attribute and the new default value; one or more attribute/value pairs can be defined. Code Example 2-20 on page 58 illustrates how this element can be modeled.

Code Example 2-20 Portion of Batch Processing File addschemaRequests.xml

```
...
<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.

ServiceConfigurationRequests Element

The *ServiceConfigurationRequests* element is reserved for future use.

DSAME XML Files

DSAME uses XML files to manage attributes that are stored in DS. It does not implement any behavior or dynamically generate any code to interpret the attributes; it can only set or get attribute values. In addition to XML files that define service attributes, DSAME also includes XML templates that can be used for batch processing. This section contains information on these types of XML files.

Internal XML Service Files

DSAME installs internal services that manage the attributes of its internal software components. The DSAME console manages the attributes for these services; in addition, DSAME provides code implementations to use them. These internal XML service files are based on the `sms.dtd`. All internal XML service files are located in `*Install_Directory*/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.
- amAuthLDAP.xml—Defines attributes for the LDAP 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.
- amClientDetection.xml—Defines attributes for the Client Detection service.
- amDomainURLAccess.xml—Defines attributes for the URL Access Policy service.
- amEntrySpecific.xml—Defines attributes for the displaying attributes on the Create, Properties and Search pages for a custom service.
- amLogging.xml—Defines attributes for the Logging service.
- amMembership.xml—Defines attributes for the Membership Authentication service.
- amNaming.xml—Defines attributes for the Naming service.
- amPlatform.xml—Defines attributes for the Platform service.
- amPolicy.xml—Defines attributes for the Policy service.
- amSession.xml—Defines attributes for the Session service.
- amUser.xml—Defines attributes for the User service.
- amWebAgent.xml—Defines attributes for the web agents.

Modifying An Internal XML Service File

Administrators can display and manage any attribute in the DSAME 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 XML service file. For example, an administrator wants to manage the `nsaccountlock` attribute; this attribute will give users the option of locking the account it defines. To manage it through DSAME, `nsaccountlock` must be described 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.

NOTE When modifying an internal XML service file, be sure to also modify the DS by extending the LDAP schema, if necessary. For more information, see "Defining A Service," on page 27.

1. Add the following code to the `SubSchema name=User` element in *`Install_Directory`*`/SUNWam/config/xml/amUser.xml`.

Code Example 2-21 `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 *`Install_Directory`*`/SUNWam/locale/en_US/amUser.properties` file with the new i18nKey tag `u13` including the text to be used for display.

Code Example 2-22 User Account Locked Example i18nKey

```
...
u13=User Account Locked
...
```

3. Remove the service `ou=iPlanetAMUserService,ou=services,dc=sun,dc=com` with `amadmin`.

   For information on the `amadmin` syntax, see "The amAdmin Command Line Executable," on page 68.

4. Reload the user service, `amUser.xml`, with `amadmin`.

   For more information on the `amadmin` syntax, see "The amAdmin Command Line Executable," on page 68.

Batch Processing XML Files

The `--data` or `-t` option of `amadmin` is used to perform batch processing using 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 singularly is to use the DSAME 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 on user objects (groups, users, roles, people containers, etc.) in the DS.

NOTE Only XML files can be used as input for the `amadmin` tool. If an administrator wants to populate the DIT in DS with user objects, or perform batch reads (gets) or deletes on the DIT, then the necessary XML input files, based on the `amadmin.dtd` or `sms.dtd`, must be written.

Batch Processing XML Templates

All of the batch processing XML files perform operations on the DIT; they create, delete, or get attribute information on user objects. The batch processing XML templates provided with DSAME include:

- `ContCreateServiceTemplate.xml`—Creates a service template for a specific container object.
- `ContModifyRequests1.xml`—Adds new attributes for a sub-container object.
- `ContModifyRequests2.xml`—Adds new attributes for a people container object.
- `ContModifyRequests3.xml`—Adds new attributes for a sub-container object.
- `ContModifyRequests4.xml`—Adds new attributes to a role object.
- `ContassignPolicyRequests.xml`—Assigns policy to a specific container object.
- `ContunassignPolicyRequests.xml`—Removes an assigned policy from a specific container object.
- `PCModifyRequests1.xml`—Adds new attributes to a people container object.
- `PCModifyUserRequests.xml`—Adds new attributes to users in a people container object.
- `RoleCreateServiceTemplates.xml`—Creates a service template for a role object.
- `RoleassignPolicyRequests.xml`—Assigns policy to a role object.
- `RolemodifyServiceTemplates.xml`—Adds new attributes to a service template for a specific role object.
- `RoleunassignPolicyRequests.xml`—Removes policy from a specific role object.
- `addChoiceValuesRequest.xml`—Adds a selection of values the user can chose from to an existing service attribute.

- `addschemaRequests.xml`—Adds a default value to an existing service attribute.
- `addserviceConfigurationRequests.xml`—This is reserved for future use.
- `createPolicyOrg.xml`—Creates policy for an organization object.
- `createRequests.xml`—Creates a multitude of objects in the DS.
- `createServiceTemplates.xml`—Creates a service template for an organization object.
- `deleteGroupRequests.xml`—Deletes all objects under a specific group container.
- `deleteOrgRequests.xml`—Deletes a multitude of objects under a specific organization.
- `deletePCRequests.xml`—Deletes a multitude of objects under a specific people container.
- `deleteServiceTemplates.xml`—Deletes a service template under a specific organization.
- `deleteserviceConfigurationRequests.xml`—This is reserved for future use.
- `getNumOfServices.xml`—Passes a listing of an organization's total number of registered services.
- `getRegisteredServices.xml`—Passes a listing of an organization's registered services.
- `getRequests.xml`—Passes information about a multitude of objects in a specific organization.
- `modifyRequests1.xml`—Adds new attributes to a number of objects in a specific organization.
- `modifyRequests2.xml`—Adds new attributes to a people container object in a specific organization.
- `modifyRequests3.xml`—Adds new attributes to a role object in a specific organization.
- `modifyServiceTemplates.xml`—Modifies existing attributes in a service registered to a specific organization.
- `modifyschemaRequests.xml`—Adds new attributes to a number of objects in a specific organization.

- `registerRequests.xml`—Registers a service to an existing organization. (This service must have been previously imported.)
- `removeChoiceValueRequests.xml`—Removes the values a user can choose from in an existing attribute in a specific service.
- `removeschemaRequests.xml`—Removes the default value of an existing attribute in a specific service.
- `unassignPolicyRequests.xml`—Removes an assigned policy from a specific organization.
- `unregisterRequests.xml`—Unregisters a service from an existing organization. (This service must have been previously imported and registered.)

These XML templates follow the structure defined by the `amAdmin.dtd`. They are located in *`Install_Directory`*`/SUNWam/samples/admin/cli/bulk-ops`.

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 the DS.

NOTE Be aware that creations of roles, groups, and organizations is a time-intensive operation.

XML Schema Files

The `ums.xml` file is the schema that defines the parameters of identity-related objects. More information on this file and how it relates to the DSAME SDK can be found in Chapter 4, "Identity Management And The SDK."

Customizing User Pages

The User entry page and what 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 DSAME console. Alternately, if an attribute is specified to be of type `User` in another XML service file, the DSAME console will display it if the service is assigned to the user. The DSAME console gets attributes to display from both, XML service files that have a defined schema attribute type of `User` and the User XML service file, `amUser.xml`. Thus, User display pages in the DSAME 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 An Internal XML Service File," on page 60.

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.

Abstract Objects and amEntrySpecific.xml

The purpose of this XML service file is to define the attributes that will display on the Create, Properties and Search pages specific to each of the DSAME *abstract objects*. Each DSAME abstract object can have its own schema definition in the `amEntrySpecific.xml` file which is based on the `sms.dtd` as described in "DSAME DTD Files," on page 33.

Abstract Objects

DSAME represents the objects it manages abstractly; in other words, an organization in DSAME does not necessarily map to an LDAP organization in the DS. The *abstract* objects are:

- organization
- organizational unit
- people container
- static group
- filtered group
- assignable dynamic group

- group container

Marker Object Classes

Abstract objects are identified in the DS by *marker* object classes that are defined in a DSAME schema and used in LDAP object entries. For example, the DS may use organizational units for their first level structure; by adding the DSAME organization marker object class, `iplanet-am-managed-org`, to the LDAP entries of these organizational units, DSAME will manage them as organizations. It is the use of marker object classes that allows DSAME to manage most directory structures, regardless of the object classes and naming attributes deployed. The marker object classes are:

- `iplanet-am-managed-filtered-group`
- `iplanet-am-managed-assignable-group`
- `iplanet-am-managed-static-group`
- `iplanet-am-managed-org`
- `iplanet-am-managed-org-unit`
- `iplanet-am-managed-people-container`
- `iplanet-am-managed-group-container`

NOTE The marker object classes are defined in the DSAME-specific LDAP schema named `95am-schema.ldif` and located in *`Install_Directory`*`/SUNWam/config/ums`. It is loaded into the DS when DSAME is installed.

amEntrySpecific.xml Schema

Each abstract object can have a schema defined in the `amEntrySpecific.xml` file. The schema defines what attributes will be displayed on the function pages used to manage abstract type objects:

- 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 abstract type object is clicked.
- Search—The Search link is in the top left frame of the DSAME console.

If a service developer wants to customize these DSAME function pages for any of the abstract objects, they would need to modify the `amEntrySpecific.xml`. For example, to display an attribute on the group page, the new attribute needs to be added to the `amEntrySpecific.xml` file. Any abstract object with customized attributes in the DS 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 2-23 is the organization attribute subschema that defines the display of an organization's Organization Status and its choice values.

Code Example 2-23 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 DSAME 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 DSAME console will not display the attribute. For additional information on the DSAME abstract objects and marker object classes, see the *iPlanet Directory Server Access Management Edition Installation and Configuration Guide*.

NOTE Note that the User service is not configured in the `amEntrySpecific.xml` file but in its own `amUser.xml` file.

The amAdmin Command Line Executable

The primary purposes of the command line executable `amadmin` is to load XML service files into the DS and to perform batch administrative tasks on the DIT. `amadmin` can be found in *`Install_Directory`*`/SUNWam/bin` and is used to:

- Load XML service files—Administrators load services into DSAME that use the XML service file format defined in the `sms.dtd`. All services must be loaded using `amadmin`; they cannot be imported through the DSAME console.

NOTE XML service files are stored in the DS as static *blobs* of XML data that is referenced by DSAME. This information is not used by the DS which only understands LDAP.

- Perform batch updates to the DIT—Administrators can perform batch updates to the DS DIT using the batch processing XML file format defined in the `amadmin.dtd`. For example, if an administrator wants to create 10 organizations, 100 people containers, 1000 users, and 100 groups, it can be done in one attempt by putting the requests in one or more batch processing XML files and loading them using `amadmin`. More information on this can be found in “DSAME DTD Files,” on page 33.

NOTE `amadmin` only supports a subset of features that the DSAME console supports and is not intended as a replacement. It is recommended that the console be used for small administrative tasks while `amadmin` is used for larger administrative tasks.

The amadmin Syntax

There are a number of structural rules that must be followed in order to use `amadmin`. The generic syntaxes for using the tool are:

- `amadmin  -u | --runasdn` *`dnname`* `-w | --password` *`password`* `[-l | --locale` *`localename`*`] [[-v | --verbose] | [-d |--debug]] -t | --data` *`xmlfile1`* `[`*`xmlfile2`* `...]`
- `amadmin -u | --runasdn` *`dnname`* `-w | --password` *`password`* `[-l | --locale` *`localename`*`] [[-v | --verbose] | [-d | --debug]] -s | --schema` *`xmlfile1`* `[`*`xmlfile2`* `...]`

- `amadmin -u | --runasdn` *`dnname`* `-w | --password` *`password`* `[-l | --locale` *`localename`*`] [[-v | --verbose] | [-d | --debug]] -r | --deleteService` *`serviceName1`* `[`*`serviceName2`* `...]`
- `amadmin -h | --help`
- `amadmin -n | --version`

NOTE Two hyphens must be entered exactly as shown in the generic syntax.

---

amadmin Options

Following are definitions of the `amadmin` command line options:

--runasdn

`--runasdn` is used to authenticate the user to the LDAP server. The argument is a value equal to that of the Distinguished Name (DN) of the user authorized to run `amadmin`; as in `--runasdn uid=amAdmin,ou=People,o=iplanet.com,o=isp`. The DN can also be formatted by inserting spaces between the domain components and double quoting the entire DN such as: `--runasdn "uid=amAdmin, ou=People, o=iplanet.com, o=isp"`.

--password

`--password` is a mandatory option and takes a value equal to that of the password of the DN specified with the `--runasdn` option.

--locale

`--locale` is an option that takes a value equal to that of the name of the locale. This option can be used for the customization of the message language. If not provided, the default locale, `en_US`, is used.

--debug

`--debug` is an option that will write messages to the `amAdmin` file created under the *`Install_Directory`*`/SUNWam/web-apps/services/debug` directory. These messages are technically-detailed but not i18n-compliant.

--verbose

`--verbose` is an option that prints to the screen the overall progress of the `amadmin` command. It does not print to a file the detailed information. Messages output to the command line are i18n- compliant.

--data

`--data` is an option that takes as its value the name of the batch processing XML file being imported. One or more XML files can be specified. This XML file can create, delete and read various directory objects as well as register and unregister services. For more information on what types of XML files can be passed to this option, see "DSAME DTD Files," on page 33.

--schema

`--schema` is an option that loads the attributes of a DSAME service into the DS. It takes as an argument an XML service file in which the service attributes are defined. This XML service file is based on the `sms.dtd`. One or more XML files can be specified.

NOTE Either the `--data` or `--schema` option must be specified, depending on whether configuring batch updates to the DIT, or loading service schema and configuration data.

--deleteService

`--deleteService` is an option for deleting a service and its schema only.

--serviceName

`--serviceName` is an option that takes a value equal to the service name which is defined under the `Service name=...` tag of an XML service file. This portion is displayed in Code Example 2-24 on page 70.

Code Example 2-24 Portion of `sampleMailService.xml`

```
<ServicesConfiguration>
    <Service name="sampleMailService" version="1.0">
        <Schema
 serviceHierarchy="/other.configuration/sampleMailService"
            i18nFileName="sampleMailService"
            i18nKey="iplanet-am-sample-mail-service-description">
...
```

--help

`--help` is an argument that displays the syntax for the `amadmin` command.

--version

`--version` is an argument that displays the utility name, product name, product version and legal notice.

SampleMailService Files

DSAME comes with the files needed to integrate a mail service into the configuration. These sample files are provided as guidelines for creating custom services and applications and illustrate how the service might be configured. The files included are:

- `sampleMailServiceSchema.ldif`—This LDAP Data Interchange Format (LDIF) file contains the LDAP schema (LDAP object classes and attribute names) for the sample mail service. The LDIF file for the service needs to be loaded into the DS using the `ldapmodify` command line tool. For more information, see the iPlanet Directory Server documentation.
- `sampleMailService.xml`—This XML service file contains the service schema and configuration parameters for the sample mail service based on the structure defined in the `sms.dtd`. It defines the mail service attributes, among them `i18Nkey` which maps to fields in the service's corresponding localization properties files.
- `sampleMailService.properties`—This localization properties file defines the object class name for the mail service profile as well as the values for the localization keys defined in `sampleMailService.xml`. The localization keys point to actual fields that display on DSAME console. For example, `i18nKey="a1"` defines a localization key in `sampleMailService.xml` file. `a1=Mail Status`, defined in `sampleMailService.properties`, tells the DSAME console to display Mail Status on the Sample Mail Service profile page in the DSAME console. For more information, see "Configuring Localization Properties," on page 32.

The files can be found in *`Install_Directory`*`/SUNWam/samples/admin/cli/sampleMailService`. These files are used throughout this chapter to illustrate service definition concepts.

NOTE	DSAME provides sample mail service files for instructional purposes only. Integrating DSAME with the iPlanet Messenger service is not supported.

Chapter 3

User Authentication With DSAME

If an organization's resources are protected by the iPlanet Directory Server Access Management Edition (DSAME), a user must submit credentials to the Authentication service in order to gain access to those resources. While DSAME provides several authentication modules, custom authentication modules may also be incorporated. This chapter explains the authentication process, its pluggable architecture and the authentication APIs. It contains the following sections:

- The Authentication Process
- Installed Authentication Services
- Custom Authentication Services
- Authentication Service Properties Files
- Authentication URL Parameters
- Authentication APIs
- Sample Authentication Service

The Authentication Process

Every organization has information and resources that need to be protected from unwanted eyes. DSAME provides secure access to these web-based applications *and* the data that it stores. Gaining access to either of these resources requires that the accessor be *validated* (given permission). DSAME can use one or more of several types of authentication methods to perform this validation.

An organization's method of authentication (their chosen *authentication service*) is defined at the root level of an organization by their administrator. When a user or application tries to access a protected resource, they are first directed to a login screen and guided through a series of one or more templates for credential gathering. Once authenticated, the user is issued an encrypted token identity and DSAME redirects them to the desired information, based on their policy set. There are two entry points which recognize that a user has not yet been validated: the DSAME console and a URL Policy Web Agent. These entry points redirect any non-validated users to the organization's authentication service.

NOTE If the authentication process fails, the user is redirected to an error page and refused entry.

Administration Console Entry

When a user (whether an organization's administrator or an end user) attempts to access DSAME's URL-based administration console, it checks the client browser for an encrypted token identity. If none is present, the user is directed to the login page of the organization's authentication service where they submit credentials for validation. Once authenticated, the user will be redirected back to the correct screen of the console, based on the roles they are assigned in their DSAME profile.

URL Policy Agent Entry

A web agent is a plug-in that resides on a web server and protects an organization's web-based resources by enforcing a user's DSAME-administered *policy*. A user's URL access policy consists of three lists of URLs: those that are not subject to policy enforcement, those that the user is denied access to and those that the user is allowed to access. When a user accesses a web-based resource by providing a URL, the web agent first checks the user's not enforced list. If a match is found there, access is allowed. If no match is found, the web agent checks the browser for an encrypted token identity. If one exists, the web agent retrieves the user's URL policy using the token information and allowed or denied access to the resource based on their policy. If there is no token identity, the user is redirected to their organization's authentication service.

NOTE URL Policy Web Agents are bundled for installation separately from the iPlanet DSAME. Additional information can be found in the iPlanet Policy Agent Pack documentation.

Client Detection

DSAME has the capability to process requests from client browsers based on a number of protocols. The client detection module determines the protocol used by the requesting client browser and retrieves the pages formatted correctly for the client type. Since any client requesting DSAME services must first be successfully authenticated, client detection is accomplished within the authentication service.

When a client's HTTP request is passed to the DSAME, it is directed to the Authentication module. Within this framework, the first step in user validation is to identify the browser type using the HTTP request. The authentication service then uses this information to retrieve the browser type's DSAME characteristics. Based on this *client data*, authentication pages are sent back to the client browser (for example, HTML or WML pages). Once the user is validated, the client type is added to the session token where it can be retrieved by other DSAME services.

Client Data

In order to recognize requesting client types, DSAME stores their identifying characteristics. This information is defined in the `iplanet-am-client-detection-client-types` property of the `amClientDetection.xml` file. The client data is separated by a pipe ("|"): `clientType=<value>|userAgent=<value>|contentType=<value>|cookieSupport=<value>|fileIdentifier=<value>|filePath=<value>`.

The fields are defined as:

- `ClientType`—an arbitrary string which uniquely identifies the client. The default is `genericHTML`.
- `UserAgent`—a search filter used to compare/match the user-agent defined in the HTTP header. The default is `Mozilla/4.0`.
- `contentType`—defines the HTTP requested content type. The default is `text/html`.
- `cookieSupport`—defines whether cookies are supported or not. The default is `true`.
- `fileIdentifier`—defines the extension of the client type files (templates and JSP). The default is `html`.
- `filePath`—defines the location of the client type files (templates and JSP files). The default is `html`.

NOTE Currently, DSAME only defines client data for HTML client types.

Installed Authentication Services

DSAME installs a number of authentication services (including the base service). This allows an administrator to choose from a variety of authentication methods with which to validate their defined organization's users. The services are:

- Core — The core service is the configuration base for all authentication method modules. It must be registered to an organization before any user can login using one of the installed authentication method modules. (In addition, the specific authentication service needs to be registered.) It allows the DSAME administrator to define default values for core authentication parameters. They can then be picked up if no overriding value is set in the specific authentication service chosen. The core values are defined in the `amAuth.xml` file.
- Anonymous — This service allows for log in without specifying a user name and password. Anonymous connections have limited access to the server and are customized by the DSAME administrator.
- Certificate — This service allows login through a personal digital certificate (PDC). iPlanet Certificate Management System (CMS) can be installed as a Certificate Authority. For more information on CMS, see the documentation set located at `http://docs.iplanet.com/docs/manuals/cms.html`
- LDAP — This service allows for authentication using LDAP bind, an operation which associates a user ID password with a particular LDAP entry.
- Membership (Self-Registration) — This service allows a new user to self-register for authentication with a login and password.
- RADIUS — This service allows for authentication using an external Remote Authentication Dial-In User Service (RADIUS) server.
- SafeWord™ — This service allows for authentication using Secure Computing's servers and tokens.
- Unix — This service allows for authentication using a user's UNIX identification and password.

Custom Authentication Services

The DSAME authentication module provides a framework that allows an organization to plug-in custom authentication services by calling the authentication APIs. The following sections provide information on how to create a custom authentication service as well as the interfaces and classes that must be implemented to run it.

Creating an Authentication Service

1. Create an XML file for the new authentication service.

   The service XML file is written so that the authentication service's *attributes* (configurable parameters) can be managed using the DSAME console. The name of this file follows the format `amAuth`*`servicename`*`.xml`; for example, `amAuthLDAP.xml` or `amAuthSafeWord.xml`. More information on writing XML files using the `sms.dtd` can be found in Chapter 2, "DSAME And XML."

2. Create a screen configuration properties file.

   A screen configuration properties file specifies the screen text that a user will see when directed to the authentication service's login page. This might include, but is not limited to, User Name and Password. The name of this file follows the format *`servicename`*`.properties`; for example, `LDAP.properties` or `SafeWord.properties`. The files are located, by default, in *`Install_Directory`*`/SUNWam/web-apps/services/WEB-INF/config/auth/default`. The directory will be different based on locale. Information on how to create the file can be found in "Configuring Screen Properties," on page 79.

3. Create a localization properties file.

   The localization properties file defines language-specific screen text for the service's attribute names. The name of the file follows the format `amAuth`*`servicename`*`.properties`; for example, `amAuthLDAP.properties`. The files are located in *`Install_Directory`*`/SUNWam/locale/`. This directory contains a sub-directory for each locale. More information on this file and how to configure it can be found in "Configuring Localization Properties," on page 81.

4. Write a Java file which implements the API `com.iplanet.authentication.spi.AuthenticationModuleFactory`.

   This API contains the method that obtains an instance of the authentication module.

5. Write a Java file which extends the API `com.iplanet.authentication.spi.AuthenticationModule`.

   This API instantiates a class of the authentication module. Certain abstract methods must be overridden.

6. Compile the application using the Java Development Kit (JDK).

   Information on how to compile a Java application can be found in the JDK documentation.

7. Modify the `amAuth.xml` file.

   Altering this file to include the new authentication service allows the pluggable architecture to recognize it.

8. Integrate the service within the DSAME authentication module by using the `amadmin` command line tool.

   Information on using `amadmin` can be found in the iPlanet Directory Server Access Management Edition Administration Guide.

Authentication Service XML Files

There are two XML files that need to be created and/or modified when creating a custom authentication service. The first file, `amAuth`*`servicename`*`.xml`, specifies the attributes that the service developer wants users and administrators to be able to configure using the DSAME console. The second file, `amAuth.xml`, defines the core authentication service.

amAuth*servicename*.xml

This file must be created for the new authentication service. Each authentication service has its own service XML file, for example, `amAuthLDAP.xml` or `amAuthSafeWord.xml`. The file specifies the attributes that a service developer wants users and administrators to be able to configure via the DSAME console. When creating it, an existing authentication service XML file can be copied and altered as needed. For information on writing a new service XML file, see Chapter 2, "DSAME And XML."

amAuth.xml

The `amAuth.xml` file defines the Core authentication service, the "parent" authentication service. After creating a new authentication service, this file must be modified in order for the authentication module to recognize the new service. This file must live in the *`Install_Directory`*`/SUNWam/config/xml` directory for all authentication modules to work. For information on modifying the `amAuth.xml` file, see Chapter 2, "DSAME And XML."

Authentication Service Properties Files

Typically, each authentication service in DSAME has two properties files: the screen properties file and the localization properties file. The screen properties file defines the screen text for the authentication service login page and the localization properties file defines locale-specific (or translated) screen text and messages for the whole service.

Configuring Screen Properties

The screen properties file specifies the screen text that a user will see when directed to that authentication service's login page. Each service's screen properties file is named using the name of the service followed by the extension `.properties`; for example, `Anonymous.properties` or `LDAP.properties`. By default, the file is in `*Install_Directory*/SUNWam/web-apps/services/WEB-INF/config/auth/default`. If the file is organization-specific, it is stored in the organization's own authentication directory `*Install_Directory*/SUNWam/web-apps/services/WEB-INF/config/*auth_orgname*`. If the files are organization and locale-specific, it is stored in the organization's locale directory inside its specific authentication directory at `*Install_Directory*/SUNWam/web-apps/services/WEB-INF/config/auth/*orgname*/*locale*`. (Information on configuring files per organization can be found in "Configuring An Organization's Screens," on page 137 of Chapter 8, "GUI Customization.") To illustrate the "The Screen Properties File Directives," on page 80, the `LDAP.properties` file has been copied below.

Code Example 3-1 LDAP.properties File

```
SCREEN
TIMEOUT 120
TEXT LDAP Authentication
TOKEN Enter UserId
PASSWORD Enter Password

SCREEN
TIMEOUT 240
TEXT Password Expiring Please Change
PASSWORD <REPLACE><BR> Enter Current Password
PASSWORD Enter New Password
PASSWORD Confirm New Password

SCREEN
TIMEOUT 120
TEXT Your password has expired. Please contact service desk to
reset your password.
```

The Screen Properties File Directives

The directives included in the configured screen properties file will depend on the requirements of the authentication method and the extent of the customization of the screen prompts. Table 3-1 discusses the directives.

Table 3-1 The Screen Properties File Directives

Directive	Description
SCREEN	Each SCREEN entry corresponds to one login page. The authentication module can set which screen is next, or it can allow the DSAME's `auth servlet` to progress through the screens sequentially.
TIMEOUT *n*	The TIMEOUT directive is used to ensure that users respond in a timely manner. If the time between when the page is sent and the user submits his response is greater than *n* seconds, a time-out page is sent.
TEXT	The TEXT directive specifies a title for the login page. Only one TEXT directive per SCREEN entry should be specified. If more than one is provided, the last one is displayed.
TOKEN *yyy*	The TOKEN directive is used to obtain the user's identification input. Within an HTML login page, it equates to the following tag: `<P><STRONG>`*yyy*`</STRONG> <INPUT TYPE="TEXT" NAME=TOKEN0>` where *yyy* is the text the user will see on the login page and INPUT specifies the input field for the user name or ID. When multiple input fields are used (such as user ID and password fields), successive numbers are appended to the name value TOKEN as in TOKEN0, TOKEN1, TOKEN2, etc. (The use of TOKEN here has no relation to a single sign-on token.)
PASSWORD *zzz*	The PASSWORD directive is used to obtain the user's password input. Within an HTML login page, it equates to the following tag: `<P><STRONG>`*zzz*`</STRONG> <INPUT TYPE="PASSWORD" NAME=TOKEN1>` where *zzz* is the text the user will see on the login page and INPUT specifies the input field for the password. When multiple input fields are used (such as user ID and password fields), successive numbers are appended to the name value TOKEN as in TOKEN0, TOKEN1, TOKEN2, etc.

Table 3-1 The Screen Properties File Directives *(Continued)*

Directive	**Description**
IMAGE *path*	The optional IMAGE directive allows for the display of a custom background image on the page where *path* equals the direct path to the displayed image.
HTML *path*	This optional HTML directive allows for the use of a custom HTML page for the authentication screens. The path attribute equals the path to the HTML file which will be displayed, overriding the HTML file dynamically generated by the authentication service's `.properties` file. For more information on customizing a HTML login page, see "Authentication URL Parameters," on page 82.
<REPLACE>	The REPLACE tag allows for the substitution of dynamic text for the static text descriptions, allowing for the dynamic generation of challenges or passwords. It is used in conjunction with the `setReplaceText()` method.

Note that the screen properties file may direct a certain number of screens to be displayed although not all of them will be. Code Example 3-1 on page 79, `LDAP.properties` has defined three screens although the last two will only be displayed under certain circumstances.

NOTE The screen properties file can also be empty. In such cases there is no login page; the credentials are specified in the URL or its part of the servlet request (as in the case of Certificate authentication).

Configuring Localization Properties

A localization properties file specifies the locale-specific screen text and localized messages that an administrator or user will see when directed to an authentication service's attribute configuration page. As an example, a portion of `amAuthLDAP.properties` is copied below. (The file is in the *`Install_Directory`*`/SUNWam/locale/` directory.) The data following the equal (=) sign in each key/value pair (displayed in English here) would be translated to a specific language as necessary and copied into the corresponding locale directory. The alphanumeric keys (`a1`, `a2`, etc.) map to fields defined, in this example, in the `amAuthLDAP.xml` service configuration file with the `i18nKey` attribute.

Note that the alphanumeric keys determine the order in which the fields are displayed on a service page in the DSAME console. The keys are taken in alphabetical and then numerical order (`a1`, `a2` is followed by `b1`, `b2` and so forth). For example, if a new attribute is added and needs to be displayed at the top of the service page, the alphanumeric key should have a value of `a1`. The second attribute should then have a value of either `a2` or `b1`, and so forth. Please note that `a10` comes before `a2`.

Code Example 3-2 Portion of `amAuthLDAP.properties`

```
...
PInvalid=Current Password Entered Is Invalid
PasswdSame=Password should not be same
PasswdMinChars=Password should be atleast 8 characters
a1=Primary LDAP Server and Port
a2=Secondary LDAP Server and Port
a3=DN to Start User Search
a4=DN for Root User bind
a5=Password for Root User Bind
a6=User Naming Attribute
a7=User Entry Search Attribute
...
```

Authentication URL Parameters

A custom HTML file can be written as an organization's login screen. This HTML file can then be dynamically generated from a value defined in the authentication service's `properties` file. For example, an organization wants its users to authenticate by entering a login ID and password in the HTML page, `MyLogin.html`. After entering the data and clicking the Submit button, the user will be taken directly to the page `http://DSAMEServer:58080/MyURL.html`. `MyLogin.html` will contain code that includes the ACTION tag as displayed in Code Example 3-3 on page 82.

Code Example 3-3 URL Parameter Code for Authentication

```
<Html>
<Head>
<Title>
My Login Form
</Title>
</Head>
<Body>
<Form  Name="login_form"  Action="/amserver/login?module=LDAP "
Method="POST">
<Input Type="TEXT" Name=TOKEN0>
```

Code Example 3-3 URL Parameter Code for Authentication *(Continued)*

```
<Input Type="PASSWORD" Name=TOKEN1>
<Input Type="SUBMIT" Name=Submit Value=Submit>
</Form>
</Body>
</Html>
```

In the source code, the HTML tag `ACTION` has the value `/amserver/login?module=LDAP`. This string can be modified by passing different name/value pairs. These pairs are:

- `goto=URL`—After successful login, DSAME redirects the user to this `URL`. It overrides the default URL. Example: `http://dsame1.red.iplanet.com:8080/amserver/login?goto=http://webserver.red.iplanet.com/webpage.html`
- `gotoOnFail=URL`—After unsuccessful login, DSAME redirects the user to this URL. Example: `http://dsame1.red.iplanet.com:8080/amserver/login?goto=http://webserver.red.iplanet.com/ForgotPassowrd.html`
- `arg=newsession`—This argument is typically used in the anonymous to authenticated user login scenario. It allows a login to destroy an existing session and perform a new login in one request. The user first hits the site with an anonymous session and then hits the Register or Login link. Example: `http://dsameserver.sun.com/amserver/login?arg=newsession`
- `module=AuthModule`—Instead of using the configured authentication modules for an organization, the authentication module is specified via this URL parameter. Example: `http://dsame1.red.iplanet.com:8080/amserver/login?module=LDAP`
- `org=OrgString`—The Authentication service figures out which organization OR sub-organization, the user is going to authenticate to based on the value of this parameter. If no `org` parameter is given, the service will use the `host:port/URI` portion of the URL.
- `page=n`—This allows applications to go directly to a specific page of a login module. For example, if a module has 4 pages and an application wants to send a user directly to page 4 it would pass page=4 in the login request. This is typically used in conjunction with custom authentication modules. For example,

```
http://dsameserver.sun.com/amserver/login?module=LDAP&TOKEN0=use
r&TOKEN1=password&page=1
http://dsame1.red.iplanet.com:8080/amserver/login?goto=
http://wevserver.red.iplanet.com/webpage.html&page=1
```

- `iPSPCookie`—To enable persistent cookies in DSAME, this parameter must be specified as true in the login URL: `iPSPCookie=true`. Persistent cookies must also be enabled in the Core Authentication service. This typically is used by portals with the *Remember my username and password* feature as it allows the user to restart their browser while retaining their session.

Authentication APIs

The authentication APIs are organized in a package called `com.iplanet.authentication.spi`. It contains the classes, interfaces and methods needed to write a customized authentication service.

NOTE The Overview page for the complete set of public Javadocs can be accessed at *`Install_Directory`*`/SUNWam/docs/index.html`.

Authentication API Overview

Each time a user attempts to access a protected resource, a new instance of the authentication Java class is created. (The reference to the class is released once the authentication session has either succeeded or failed.) When an authentication session is invoked, one login page is sent to the browser for each screen directive defined in the screen properties file although not all screens will need to be displayed. The first directive would send a login page asking the user to enter a user identification and a password. When the user submits the information, the `validate()` method is called. The module gets the information tokens, validates and returns them. If applicable, a second screen is sent and the `validate()` method is called again. (In the `LDAP.properties` Code Example 3-1 on page 79, a second screen would be sent only to a user whose current password is expiring.) When multiple screens are sent to the user, the tokens from a previous screen can be retrieved by using the `getTokenForState` methods. (Each screen is referred to as a *state*.) The authentication module keeps all tokens from previous states until authentication is complete.

Naming Conventions

The following naming convention is recommended when creating the custom authentication service. If the new authentication service class is named *servicename*`.java`, the authentication module factory class should be named *servicename*`AuthenticationModuleFactory.java`. In addition, the use of upper and lower case letters should be consistent. If the new authentication service class is named `NewAuth.java`, the authentication module factory class should be named `NewAuthAuthenticationModuleFactory.java`.

AuthenticationModuleFactory Interface

The `AuthenticationModuleFactory` interface must be implemented for each custom authentication module. This top-level class contains the `newAuthenticationModule()` method which creates a new instance of the `AuthenticationModule` class. This new instance defines the authentication module being customized.

AuthenticationModule Class

The AuthenticationModule class extends the Authenticator class which defines basic methods used in the authentication service. The AuthenticationModule class contains more detailed methods. The instance of the AuthenticationModule class must override the `validate()`, `init()`, and `getUserTokenId()` methods.

- The `init()` method should be used if the class has any specific initialization requirements such as loading a JNI library. `init()` is called once for each instance of the class. Once a login session is completed, the reference to the class is released.
- The `validate()` method is called for each authentication page specified in the screen properties file and validates the entered credentials and thus, the user. At the point of authentication failure, it throws an `AuthenticationException`. The reason for failure can be an argument to the exception and will be logged in the DSAME authentication log.
- The `getUserTokenId()` method is called once at the end of a successful authentication session. A login session is deemed successful when all pages in the screen properties 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 DSAME environment.

NOTE	If the instance of DSAME participates in the User Lockout feature, the `validate()` method throws an `InvalidPasswordException` after *n* attempts at login. In order to support this feature, the `getUserTokenId()` method should be set to return the user ID before the exception is thrown.

LoginWorkerFactory

The `LoginWorkerFactory` interface must be implemented for each custom *non-HTML* authentication module. This top-level class contains the `newLoginWorker()` method which creates a new instance of the `LoginWorker` class. The new instance generates the UI for the authentication module being customized.

NOTE	Any static data or reference to any static data in the authentication module must be thread-safe.

Sample Authentication Service

A sample authentication program has been provided in the directory, `*Install_Directory*/SUNWam/samples/authentication/providers`. It includes the following files:

- `AuthenticationSample.jar`
- `AuthenticationSample.java`
- `AuthenticationSample.properties`
- `AuthenticationSampleAuthenticationModuleFactory.java`
- `Readme.html`

The `Readme.html` file explains how to compile, deploy and run the Authentication Sample program. It is copied below.

Authentication Sample: Readme.html

The `Readme.html` file explains how to compile, deploy and run the Authentication Sample program.

Steps to compile the Authentication Sample program

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 same directory (*`Install_Directory`*`/SUNWam/samples/authentication/providers`) as the Authentication Sample program files.

- `JAVA_HOME` - Set this variable to your installation of JDK. The JDK should be newer than JDK 1.2.2.
- `CLASSPATH` - Modify the /opt to the base of your installation. *`Install_Directory`*`/SUNWam/web-apps/services/WEB-INF/lib` directory.
- `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. Go to the *`Install_Directory`*`/SUNWam/samples/authentication/providers` directory and run `gmake`.

Steps to deploy the Authentication Sample program

1. Copy `AuthenticationSample.jar` from `JAR_DIR` to *`Install_Directory`*/SUNWam/web-apps/services/WEB-INF/lib".

2. Copy `AuthenticationSample.properties` from *`Install_Directory`*`/SUNWam/samples/authentication/providers` to *`Install_Directory`*`/SUNWam/web-apps/services/WEB-INF/config/auth/default`.

NOTE	The properties file name should be the same as the Authentication Sample module name.

3. Modify *`Install_Directory`*`/SUNWam/web-apps/services/WEB-INF/config/xml/amAuth.xml` to include the Authentication Sample in the Authentication menu choices and in the Authenticator's list (in Admin Console) as follows:

Code Example 3-4 `amAuth.xml` After Modification

```
<AttributeSchema name="iplanet-am-auth-menu"
              type="multiple_choice"
              syntax="string"
              i18nKey="a1">
              <ChoiceValues>
                  <Value>LDAP</Value>
                  <Value>Radius</Value>
                  <Value>Membership</Value>
                  <Value>Anonymous</Value>
                  <Value>Cert</Value>
                 <Value>AuthenticationSample</Value>
              </ChoiceValues>
              <DefaultValues>
                  <Value>LDAP</Value>
             </DefaultValues>
          </AttributeSchema>
          .....
          .....
          .....
          <AttributeSchema name="iplanet-am-auth-authenticators"
              type="list"
              syntax="string"
              i18nKey="a17">
              <DefaultValues>
 <Value>com.iplanet.authentication.modules.radius.Radius</Value>
 <Value>com.iplanet.authentication.modules.ldap.LDAP</Value>
<Value>com.iplanet.authentication.modules.membership.Membership<
/Value>
<Value>com.iplanet.authentication.modules.anonymous.Anonymous</V
alue>
 <Value>com.iplanet.authentication.modules.cert.Cert</Value>
<Value>com.iplanet.authentication.modules.application.Applicatio
n</Value>
<Value>com.iplanet.am.samples.authentication.providers.Authentic
ationSample</Value>
              </DefaultValues>
 </AttributeSchema>
```

4. Make a backup copy `amAuth.xml`.
5. Delete `iPlanetAMAuthService` entry and then import (the modified) `amAuth.xml` using `amadmin`.
   a. `cd <install-root>/SUNWam/bin`
   b. `./amadmin --runAsDN uid=amAdmin,ou=People,`*`<default_org>`*`,`*`<root_suffix>`*` --password `*`<password>`*` --deleteService iPlanetAMAuthService`

   c. ./amadmin --runAsDN uid=amAdmin,ou=People,<*default_org*>,<*root_suffix*> --password <*password*> --schema amAuth.xml

6. Add the AuthenticationSample.jar file path to the Web server JVM classpath:

   a. cd *Install_Directory*/SUNWam/servers/https-<host>.<domain>/config

   b. Modify jvm12.conf to add *Install_Directory*/SUNWam/web-apps/services/WEB-INF/lib/AuthenticationSample.jar path to the JVM classpath.

Steps to run the Authentication Sample program

1. Restart DSAME server *Install_Directory*/SUNWam/web-apps/services/WEB-INF/bin/amserver start.

2. Log in to the DSAME console by entering the URL http://<host>.<domain>:<port>/<Deploy-URI>/console.

3. Select the User Management view.

4. Select your organization and select services from the Show menu.

5. Click on the DSAME Core Authentication properties icon.

6. Add the Authentication Sample class in Pluggable Auth Module Classes.

7. Select AuthenticationSample from Authentication Menu.

8. Click Submit to save changes and log out.

9. Enter the URL http://<host>.<domain>:<port>/<Deploy-URI>/login and select AuthenticationSample from Authentication Menu *OR* enter the URL http://<host>.<domain>:<port>/<Deploy-URI>/login?module=AuthenticationSample.

Chapter 4

Identity Management And The SDK

The Identity Management module of DSAME contains XML templates and application programming interfaces (APIs) that can provide functionality to, among other operations, create, delete and managing identity entries in the directory. This chapter offers information on these public API. It contains the following sections:

- Overview
- Management Of Identity-Related Objects
- DSAME SDK
- The SDK And Cache

Overview

The Identity Management module of DSAME provides interfaces for creating and managing identity-related objects in the iPlanet Directory Server (DS). The management functions that can be performed include the creation and deletion of the specific objects as well as the ability to get, add, modify, or remove attributes of these objects. The interfaces provided for this feature are a Java SDK to embed the management functions with applications or services, and a set of configuration parameters (defined in the `ums.xml`). The following sections describe the configuration Templates and the DSAME SDK.

Management Of Identity-Related Objects

The `ums.xml` provides a set of configuration parameters, known as Templates, that contain LDAP configuration information for identity-related objects. (It can be found in the *`Install_Directory`*`/SUNWam/config/ums` directory.) The identity-related objects are:

- Users
- Groups
- Organizations
- Roles
- Organization Units
- Group Containers
- People Containers

The templates are used by the DSAME SDK for the creation of these objects in the DS, as well as the dynamic generation of the object's roles and the construction of object searches. (These templates can be modified by administrators to alter the behavior of the Java interfaces.) Using these templates and the LDIF schema, parameters are configured for all identity-related objects.

When DSAME is installed, the `ums.xml` file is stored in the DS as the *DAI* service. (DAI is a service in DSAME whose configuration is not made available through the DSAME console.) The DSAME SDK gets the configuration information from this node when it is being asked to create an identity-related object, generate a role or perform a search. Any attribute specified in the `ums.xml` can be set for a created object.

NOTE `ums.xml` has template definitions for the various directory entries created by the SDK. If it is modified and reloaded with those modifications, there would be inconsistencies between the new entries created and the older ones. Hence, modifications to this file are not recommended unless DSAME is being installed fresh.

Structure of ums.xml

The `ums.xml` defines three templates: Structure, Creation and Search. Structure templates define the DS DIT attributes for the object. Creation templates define an LDAP template for the object being created. Search templates define guidelines for performing searches using LDAP. These concepts are discussed in depth below.

Structure Templates

Structure templates define the form a DSAME object will take in the DS DIT. This conforms to where the object is located within the DIT; the objects are strictly LDAP entries. There are six attributes that need to be defined for each subschema.

- `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 entry type of the object. For example, an organization object has *o=org* as its name.
- `childNode`—This attribute specifies the child nodes that will be created in tandem with the object.
- `template`—This attribute specifies the Creation template used to create this object.
- `filter`—This attribute specifies a filter that will be used to identify the object.
- `priority`—This attribute is defined as 0.

Creation Templates

Every entry that DSAME creates has a corresponding creation template which defines the LDAP schema for the object being created. It specifies what object classes and attributes are mandatory or optional and what default values, if any, should be set. This conforms to the actual LDAP entry in the DS. There are six attributes that need to be defined for each subschema.

- `name`—This attribute defines the name of the object the template will create. It is also the name of the template itself.
- `javaclass`—This attribute defines the name of the Java class used to instantiate the object.
- `required`—This attribute defines the required LDAP attributes for the object.
- `optional`—This attribute defines the optional LDAP attributes for the object.
- `validated`—This attribute is reserved for future use.

- `namingattribute`—This attribute specifies the LDAP entry type.

Search Templates

Search templates are used to define how DSAME searches are performed in the DS. This template defines a default search filter and the returning attributes in a search. For example, a search filter is constructed which defines and specifies which attributes and values are to be retrieved from the DS.

- `name`—This attribute defines the name of the search template.
- `searchfilter`—This attribute defines the LDAP search filter.
- `attrs`—This attribute specifies the LDAP attributes that need to be returned.

Modifying ums.xml

In addition to modifying an XML service file, any new LDAP attributes or object classes must be added to the `ums.xml` file in order for them to be recognized by DSAME. 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 `employee_id` attribute, the `ums.xml` file does not need to be modified because this attribute already exists in the `inetorgperson` object class. Generally, as in the example, the `ums.xml` file does not need to be modified. The only circumstances where this file would need to be modified are:

- if DSAME 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 *iPlanet Directory Server Access Management Edition Installation and Configuration Guide*.

CAUTION It is highly recommended that the `ums.xml` configuration file is duplicated before any modifications are made.

Adding Custom Object Classes

If a service developer wanted to add new or customized object classes to DS for DSAME's use, they would need to modify the templates in the `ums.xml` file to include them. Then, to manage them from the DSAME console, these new object classes and attributes have to be modelled in the XML service file format and imported into DSAME using the procedures described in this chapter.

DSAME SDK

The DSAME SDK contains APIs for identity management. These interfaces can be used by developers to integrate management functions into external applications or services to be managed by the DSAME. The following sections describe the Java classes.

NOTE The public Javadocs can be accessed through *`Install_Directory`*`/SUNWam/docs/index.html`.

Identity Management APIs

The Identity Management APIs provide the means 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 the DS.

AMConstants

`AMConstants` is the base interface for all identity-related objects. It is used to define the scope of a search of the DS. It can search for a specific object, a particular level of the DIT or an attribute.

AMObject

`AMObject` provides basic methods to manage identity-related objects. Since this is a generic class, it does not have any Templates 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 name of the structural template used by this class is *Organization*; the name of the creation template used is *BasicOrganization*, and the name of the search template is *BasicOrganizationSearch*.

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 name of the structural template used by this class is *OrganizationalUnit*; the name of the creation template used is *BasicOrganizationalUnit*, and the name of 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 name of the structural template used by this class is *PeopleContainer*; the name of the creation template used is *BasicPeopleContainer,* and the search template is *BasicPeopleContainerSearch*.

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 name of the structural template used by this class is *GroupContainer*; the name of the creation template used is *BasicGroupContainer*, and the search template is *BasicGroupContainerSearch*.

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.

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.

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.

AMAssignableDynamicGroup

The `AMAssignableDynamicGroup` interface provides the methods used to manage assignable 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 *BasicAssignableDynamicGroup*; and the search template used is named *BasicAssignableDynamicGroupSearch*. It does not have a pre-defined structural template.

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 named *BasicManagedRole*; and the search template used is named *BasicManagedRoleSearch*. It does not have a pre-defined structural template.

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 named *BasicUser*; and the search template used is named *BasicUserSearch*. It does not have a pre-defined structural template.

AMTemplate

The `AMTemplate` interface represents a service template associated with a `AMObject`. DSAME distinguishes between virtual and entry attributes. Per iPlanet Directory Server (DS) terminology, 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. Entry attributes are non-inherited attributes.

NOTE More information on virtual attributes can be found in "Virtual Attribute," on page 135 of Chapter 8, "iPlanet Directory Server And DSAME."

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. There are three types of templates: `POLICY_TEMPLATE`, `DYNAMIC_TEMPLATE` and `ORGANIZATION_TEMPLATE`. `POLICY_TEMPLATE` and `DYNAMIC_TEMPLATE` are implemented using CoS Templates; `ORGANIZATION_TEMPLATE` does not have virtual attributes.

Template Priority

When any 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), conflicts between such templates are resolved by the use of template priorities. In this priority scheme, zero is the highest possible priority with the lower priorities extending towards finity. Templates with higher priorities will be favored over and to the exclusion of templates with lower priorities. Templates which do not have an explicitly assigned priority are considered to have the lowest priority possible, or no priority. 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 undefined, but does not exclude the possibility that a value will be returned, however arbitrarily chosen.

AMStoreConnection

The `AMStoreConnection` class represents a connection to the DSAME data store. It controls and manages access to the DSAME data store by providing methods to create, remove and get different types of identity-related objects. A SSO Token is required in order to instantiate a `AMStoreConnection` object.

Sample Code

Following are code samples using the DSAME SDK.

Create Organization

The following code sample creates a new organization with one user by opening a connection to the DS data store with `AMStoreConnection`. A new top organization (`newtoporg.com`) is then created with its own attributes. User John Smith is then created as a member of the new organization.

Code Example 4-1 Create a new organization and one user

```
...
     // instantiate a store connector from SSO Token
     AMStoreConnection  amsc = new AMStoreConnection(ssoToken);
     // create a new top level organization without non-default
attributes
     AMOrganization org =
amsc.createTopOrganization("newtoporg.com", new HashMap());
     // set attribute for the newly created organization
     org.setStringAttribute("description", "organization
description");
     // save new attribute to the organization object
     org.store();

     // create new user "john" with "cn", "sn" attribute
     // Map to hold all users to be created, key is the string
value for user naming attribute,
     // value is a Map which contains all the initial values for
the user
     Map usersMap = new HashMap();
     // Map to hold attributes for the user
     Map attrsMap = new HashMap();
     // set cn = John Smith
     Set values = new HashSet();
     values.add("John Smith");
     attrsMap.put("cn", values);
     // set sn = Smith
     values = new HashSet();
     values.add("Smith");
     attrsMap.put("sn",  values);
     // set put user john in the usersMap with "cn" & "sn"
specified above
     usersMap.put("john", attrsMap);
     // create user john in the organization
     Set users = org.createUsers(usersMap);
...
```

Retrieve Templates

The following code sample retrieves a service's dynamic templates by opening a connection to the DS data store with `AMStoreConnection`. It retrieves a service's dynamic template by defining the DN of the top organization (`toporg.com`) as well as the specific string attribute of the specific service to be retrieved.

Code Example 4-2 Retrieve a 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("o=toporg.com,o=isp");
     // 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
");
     ...
```

The SDK And Cache

Caching in the DSAME SDK is for storing all `AMObject` attributes (i.e., attributes of identity-related objects) that are retrieved from iDS. The cache does not hold `AMObject` directly. All attributes retrieved from the DS using the interface methods `AMObject.getAttribute(String name)`, `AMObject.getAttributes(setAttributeNames)` or `AMObject.getAttributes()` will be cached.

Cache Properties

The following cache properties can be configured by accessing the `AMConfig.properties` file. They are:

- `com.iplanet.services.stats.state`—Depending on whether this property is set to `file` or `console`, the cache statistics will be printed to either a `amSDKStats` file or the DSAME console.

- `com.iplanet.services.stats.directory`—The value of this property is the directory in which the `amSDKStats` file is created.
- `com.iplanet.am.statsInterval`—The interval at which cache statistics are printed can be specified as the value of this property. It indicates the number of seconds after which the stats will be printed. For example, a value of 3600 would cause the cache statistics to be printed after 3600 seconds. This will be used only if `com.iplanet.services.stats.state` is set to `file` or `console`.

Code Example 4-3 on page 101 is an example of how the statistics will be formatted.

Code Example 4-3 Format of Recorded Statistics

```
01/15/2002 09:12:35:751 AM PST: Thread[Thread-47,5,main]
SDK Cache Statistics
------------------------------
Interval:  <number of get requests during the specified interval>
Hits during interval:  <number of hits during this interval>
Hit ratio for this interval:  <hit ratio for the interval>
Total number of requests: <overall (total) number of get requests
since server re-start>
Total number of Hits: <overall (total) number of hits since
server re-start>
Overall Hit ratio:  <overall hit ratio since server re-start>
```

Chapter 5

Single Sign-On With DSAME

The iPlanet Directory Server Access Management Edition (DSAME) provides a single sign-on (SSO) solution that enables a user to authenticate once in order to access multiple applications and 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 solution, how it works and the SSO APIs. It contains the following sections:

- The Single Sign-On Process
- Cross-Domain Support For SSO
- SSO APIs
- Sample SSO Java Files
- Multi-JVM Support

The Single Sign-On Process

DSAME uses access control mechanisms to protect an organization's proprietary data and web resources. A user wanting to access these protected resources must first pass validating credentials through the Authentication service. A successful authentication gives the user authorization to access protected resources, based on their assigned policies or other such mechanism. If a user wants access to *several* resources protected by DSAME, the SSO (or *Session*) service provides proof of authentication to those resources so there is no need to re-authenticate. These different domains generally have common users who need to generate access to their services in a single user session.

Contacting A Web Agent

When a user, using a web browser, attempts to access a protected resource, the URL Policy Agent intercepts the request. Web agents police the web or application server on which the protected resource lives. Web agents enforce three types of policy: those URLs that can be accessed by the user, those URLs for which the user is denied access and those that are not subject to policy enforcement.

NOTE URL Policy Agents are bundled for installation separately from the iPlanet DSAME. Additional information can be found in the iPlanet Policy Agent Pack documentation.

When the web agent intercepts the user's request, it checks to see if the requested URL is not subject to policy enforcement. If there is a match, the agent allows immediate access. If there is no match, the agent understands that the URL is subject to policy enforcement and inspects the request further to see if a user session identifier, or *token*, exists. If none exists, the request is passed to the DSAME server where it contacts both, the Session service to create a user *token* and the Authentication service to verify the user.

Creating A Session

Before a user's credentials can be authenticated, a *token* is generated using the *Session* service. Each token contains a randomly-generated DSAME session identifier and ultimately represents an authenticated user. Once created, the Authentication service inserts the token into a cookie and assigns it to the client browser. At the same time the token is assigned, a HTML login page is returned to the user based upon their organization's method of authentication (LDAP, RADIUS, Unix, etc.).

NOTE The session token, at this point, is in an *invalid* state and will remain in one until the user has completed authentication.

Providing User Credentials

The user, having received the correct login page as well as a session token, fills in the appropriate user ID and password based on the login page returned. After the user enters 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 from the token and the session state is set to *valid*. The user can now be redirected to the URL they were attempting to access.

Cookies and Tokens

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 it is generated by. It does not imply that the user is authenticated. Cookies are domain-specific; for example, a cookie generated by `thisdomain.com` cannot be used in another domain such as `thatdomain.com`. In a DSAME implementation, the cookie is generated by DSAME's Session service and set by the Authentication service.

A *token* is generated by DSAME's Session service and inserted into a cookie. The token is generated using a secure random number generator and contains DSAME-specific session information. When a protected resource is accessed, the user is validated by the Authentication service and a `SSOtoken` is created.

Cross-Domain Support For SSO

DSAME supports cross-domain SSO. A user authenticated to DSAME in one domain can access resources protected by that same DSAME server in another domain. For example, the DSAME instance for `DomainA` is the authentication provider. A user authenticates to DSAME in `DomainA` and, after authentication, the token is set for `DomainA`. `ServerB` is protected by a web agent talking to a DSAME server in `DomainB`. The DSAME server in `DomainB` recognizes the `DomainA` server as the authentication provider.

If `UserA` accesses a resource on `ServerB` after authenticating to DSAME in `DomainA`, the web agent at `DomainB` checks to see if the request has a SSO token and finds that there is no `DomainB` token in the request. In a cross-domain SSO scenario, the agent will redirect the user to the URL of the cross-domain component running with the DSAME server in `DomainB`. This component redirects the request to the cross-domain component on `DomainA` since the DSAME at `DomainA` is the authentication provider. This request receives the SSO token set by DSAME in `DomainA` in the cookie header. The component at `DomainA` will send a response back to the component in `DomainB` with a SSO token. The `DomainB` component validates the SSO token from `DomainA` and creates the SSO token for the user in `DomainB`. This process sets a cookie for the user in `DomainB`. who is given access to the requested resource only if their policy grants authorization to access it.

If a user accesses a resource directly at `DomainB` without authenticating at `DomainA`, the user is redirected to authentication at `DomainA`. If the authentication is successful, the SSO token is sent to `DomainB` from `DomainA`. The `ServerB` validates the SSO token with `DomainA`, creates it for `DomainB` and redirects the user to the original requested resource.

Enabling Cross-Domain Single Sign-On

To enable cross-domain SSO, the administrator needs to configure two different SSO components. They are the `Cross Domain Controller` and the `SSO Component`. The `Cross Domain Controller` component comes bundled, and is installed, with DSAME. The `SSO Component` is a domain agent that needs to be installed separately onto all participating DNS domains.

NOTE The system administrator can choose to not enable the cross-domain feature; in this case the SSO component would function within the context of a single domain.

Cross Domain Controller

The `Cross Domain Controller` (CDC) is associated with the DSAME server that is protecting a specific domain. It redirects a request to either the Authentication service or to the `SSO Component`. When a HTTP request comes into the CDC and no SSO token information is found, the request is redirected to the Authentication Service. If a SSO token is found for another domain, the request is redirected to the `SSO Component` with the appropriate session information appended to the query string.

NOTE The CDC is installed when the command line tool `aminstall` is run to install the DSAME application. For more information, see the *iPlanet Directory Server Access Management Edition Installation and Configuration Guide.*

SSO Component

The `SSO Component` is deployed in each DSAME-protected domain. When a user attempts to access a resource, the URL is intercepted by the web agent as discussed in "Contacting A Web Agent," on page 104. If no SSO token is found, the request is redirected to the `SSO Component` in the domain where the resource exists. The `SSO Component` searches the query string again for the SSO token. As no token is found, the request is redirected to the `Cross Domain Controller` associated with the DSAME server that protects this resource. From this point, the authentication process will be followed.

NOTE If a SSO token is found by the web agent when the request is made, the `SSO Component` would not receive the request as the web agent would take the course of validating the token as described in Chapter 3, "User Authentication With DSAME."

Configuring For Cross-Domain SSO

The SSO components need to be enabled in order to allow the cross-domain SSO function to work. Assuming a single DSAME instance:

1. Run `aminstall` to install DSAME.

   This will install the DSAME application as well as the CDC component and other internal services. The default CDC service URL, after installation, is `http(s)://DSAME-HOST:PORT/amserver/cdcservlet`.

2. Run `aminstall` again and choose to install the Cross-Domain Support option.

   All participating DNS domains need to have an instance of the SSO component installed in their domain. After running this installation option, a `cdsso` directory is created in *`Install_Directory`*`/SUNWam/web-apps`. The default SSO Component service URL, after installation, is `http(s)://DOMAIN-CDSSO-HOST:PORT/uri/cdssoservlet`.

NOTE Install the SSO Component on any web server with host services that need to be protected in all participating DNS domains.

---

3. Edit the `com.iplanet.services.cdsso.cookiedomain` property in the `cdsso.properties` file found in the *`Install_Directory`*`/SUNWam/web-apps/cdsso/WEB-INF/classes` directory.

Set the `com.iplanet.services.cdsso.cookiedomain` property to the domain name which hosts the SSO component. For example, `com.iplanet.services.cdsso.cookiedomain =.sales.com`, if the SSO component is hosted in a sun.com domain. Code Example 5-1 is copied from the file itself.

Code Example 5-1 Portion of `cdsso.properties` file

```
...
/*
* The following keys will be used for Cross Domain SSO support.
* The user if needs cross doamin sso support should change
*"com.iplanet.services.cdsso.CDCURL" property to point to the
* cdcservlet running with the DSAME instance
* "com.iplanet.services.cdsso.cookiedomain" property should
* specify a comma separated list of domains for which the cdsso
* servlet will set a SSOToken.
* Ex:com.iplanet.services.cdsso.cookiedomain=.sales.com,
.eng.com,.marketing.com
*/

com.iplanet.services.cdsso.CDCURL=http://rays.india.sun.com:8080
/amserver/cdcservlet
com.iplanet.services.cdsso.cookiedomain=.sales.com
/*
...
```

4. Edit three properties in each web agent's `AMagent.properties` file.

   - Change the value of `com.iplanet.am.policy.agents.url.authLoginUrl` so it points to the component's domain's SSO service URL. For example, `com.iplanet.am.policy.agents.url.authLoginUrl=http(s)://DOMAIN-CDSSO-HOST:PORT/uri/cdssoservlet`. Code Example 5-2 illustrates where this property can be found.

Code Example 5-2 Second portion of CDSSO `AMConfig.properties` file

```
...
/*To enable cross domain sso support
"com.iplanet.am.policy.agents.url.authLoginUrl" needs to be
*changed to point to the cdsso servlet
*instead of the login servlet*/
/*com.iplanet.am.policy.agents.url.authLoginUrl=PROTO://HOST:POR
T/DEPLOY_URI/cdssoservlet*/

com.iplanet.am.policy.agents.url.authLoginUrl=SERVER_PROTO://SER
VER_HOST:SERVER_PORTSERVER_DEPLOY_URI/login
...
```

- Add the SSO service URL to both the component's local and remote *not enforced* list. Code Example 5-3 displays the portion of the file where these properties are defined.

Code Example 5-3 Third portion of CDSSO `AMConfig.properties` file

```
...
/*If cross domain sso support is enabled notenforcedlist should
*be edited to add cdsso servlet URL in it
*com.iplanet.am.policy.agents.url.notenforcedlist.local=
*PROTO://HOST:PORT/DEPLOY_URI/cdssoservlet */
com.iplanet.am.policy.agents.url.notenforcedlist.local=SERVER_PR
OTO://SERVER_HOST:SERVER_PORTSERVER_DEPLOY_URI/console*,
SERVER_PROTO://SERVER_HOST:SERVER_PORTSERVER_DEPLOY_URI/login*,
SERVER_PROTO://SERVER_HOST:SERVER_PORTSERVER_DEPLOY_URI/images/*
,
SERVER_PROTO://SERVER_HOST:SERVER_PORTSERVER_DEPLOY_URI/logout,
SERVER_PROTO://SERVER_HOST:SERVER_PORTSERVER_DEPLOY_URI/namingse
rvice,
SERVER_PROTO://SERVER_HOST:SERVER_PORTSERVER_DEPLOY_URI/sessions
ervice,
SERVER_PROTO://SERVER_HOST:SERVER_PORTSERVER_DEPLOY_URI/loggings
ervice,
SERVER_PROTO://SERVER_HOST:SERVER_PORTSERVER_DEPLOY_URI/profiles
ervice,
*AGENT_DEPLOY_URI/html/URLAccessDenied.html,
SERVER_PROTO://SERVER_HOST:SERVER_PORTSERVER_DEPLOY_URI/admin/*,
SERVER_PROTO://SERVER_HOST:SERVER_PORTSERVER_DEPLOY_URI/docs*,
SERVER_PROTO://SERVER_HOST:SERVER_PORTSERVER_DEPLOY_URI/index.ht
ml,
SERVER_PROTO://SERVER_HOST:SERVER_PORTCONSOLE_DEPLOY_URI/*,
SERVER_PROTO://SERVER_HOST:SERVER_PORTSERVER_DEPLOY_URI/GetHttpS
ession
...
/*If cross domain sso support is enabled notenfocelist should be
edited to add cdsso servlet URL in it
com.iplanet.am.policy.agents.url.notenforcedlist.remote=PROTO://
HOST:PORT/DEPLOY_URI/cdssoservlet*
*/
com.iplanet.am.policy.agents.url.notenforcedlist.remote=*AGENT_D
EPLOY_URI/html/URLAccessDenied.html
```

This instance of DSAME and all its participating DNS domains are now cross-domain SSO enabled.

NOTE The cross-domain SSO solution assumes a single DSAME instance; therefore all user and policy information needs to be centralized in that instance. Multiple DSAME instances are allowed if they are all in the same domain.

SSO APIs

The SSO solution provides Java API to allow external applications to participate in the SSO functionality. All DSAME's services (except for Authentication) need a valid SSO token to process a HTTP request. External applications wishing to use the SSO functionality must use the SSO token to validate the user's identity. With the SSO API, an external application can get the token and, in turn, the identity of a user and related authentication information. Once a user is authenticated, this information is used to determine whether or not to provide access to the requested resource based on the validated user's policy. The SSO API can also be used to create or destroy a SSO token, to check the token's validity or to listen for token *events*. (An event might be a token timing out because the user has reached the token's maximum time limit.)

Non-Web-Based Applications

DSAME provides the SSO component 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 DSAME cookie value and pass it into the SSO client methods to get to the SSO token. The method used for this process is application-specific.

2. Command line applications, such as `amadmin`, can be used. In this case, SSO tokens can be created to access the DS directly. There is no session created making the DSAME access is valid only within that process or VM.

API Overview

The primary purpose of the SSO API is to allow any service or application to make use of the SSO functionality. They are provided for the implementation of a SSO solution in external applications. Using these APIs, the identity of the user and related authentication information can be called. The application then uses this information to determine whether to provide user access to a protected resource. The SSO client applications get the information from the SSO token. For example, assume a user authenticates to `http://www.DomainA.com/Store` successfully and later tries to access `http://www.DomainB.com/UpdateInfo`. Rather than having the application authenticate the user again, it can use the API to determine if the user is already authenticated. If the methods indicate that the user is valid and has already been authenticated, access to this page can be given without the user authenticating again. Otherwise, the user is prompted to authenticate again.

Each time a user attempts to access a protected application, the application needs to verify their validity. Generally, the SSO component generates a SSO token for a user once the user is authenticated. After generation, the token is carried with the user as the user moves around the web. When the user attempts to access an application or service that is *SSO-enabled*, this token is used for user validation. Specifically, an instance of the `SSOTokenManager` class is created to allow access to the `createSSOToken`, `destroyToken` and `isValidToken` methods. An instance of the `SSOToken` class is then called; it contains the session information. Between the two, an application can determine if the user is authenticated. Another way to use the API is to invoke the `SSOTokenListener` interface which notifies the application when a token has become *invalid* in order for the application to terminate its access.

NOTE For more information on the SSO APIs, the public Javadocs can be accessed through *`Install_Directory`*`/SUNWam/docs/index.html`.

SSOTokenManager Class

The `SSOTokenManager` class must be implemented to create one instance per token. It contains the three methods needed to create, get, validate and destroy SSO tokens. The `createSSOToken()` method is called to create a session token. It contains methods for doing this using the command line or through the internet. The `destroyToken()` method is called to delete a token when its session has ended. The `isValidToken()` and `validateToken()` methods are called in tandem to verify the authenticity of a token.

NOTE SSOTokenManager is a final class and a singleton. SSOToken and SSOTokenID are Java interfaces. Additionally, SSOTokenListener and SSOTokenEvent are provided to support notification when SSO tokens are invalidated.

Sample SSOTokenManager Code

The SSOTokenManager class can be used in the following way to determine if a user is authenticated:

Code Example 5-4 Sample SSOTokenManager Code

```
try {
    /* create the sso token from http request */
    SSOTokenManager manager =  SSOTokenManager.getInstance();

   /* The request here is the HttpServletRequest. */
    SSOToken token = manager.createSSOToken(request);

   /* use isValid to method to check if the token is valid or not
    * this method returns true for valid token, false otherwise*/
   if (token.isValid()) {

 /* user 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,
* by some means.If access can be given based on this
* further check on user information is not necessary.
          */

        /* let us get some user information */
        String host = token.getHostName();
       java.security.Principal principal = token.getPrincipal();
        String authType = token.getAuthType();
        int level = token.getAuthLevel();

         ..........

    } else {
        /* token is not valid, redirect the user login page */
    }
...
```

SSO Implementations

The `SSOTokenManager` maintains a configuration database of valid implementations for `SSOProvider`, `SSOToken` and `SSOTokenID`. A request to `SSOTokenManager` gets delegated to the `SSOProvider`. Hence, the `SSOProvider` performs the bulk of the function of `SSOTokenManager`. The `SSOToken` is the SSO token that contains the crucial information about the token, and `SSOTokenID` is a string representation of SSO token. Although `SSOTokenManager` could support multiple and disparate providers, the only valid SSO provider is `SSOProvider`.

Additional Classes

The following classes can be used to implement customized SSO functionality in an application that does not use the default `SSOProvider` provided.

SSOToken

The `SSOToken` class represents a "single sign-on" token and contains information like the user validation, the authentication method, the host name of the client browser that sent the request, and session information (maximum session time, maximum session idle time, session idle time, etc.). Code Example 5-4 on page 112 also makes use of the `SSOToken` interface.

SSOTokenEvent

The `SSOTokenEvent` class represents a token event. An event is, for instance, when a token becomes invalid due to idle time-out or hitting a time limit maximum. A token is granted when a change in the state of the token, like those mentioned, occurs. An application must come to know of events in order to terminate access to the application for a user whose token has become invalid. The `SSOTokenListener` class would need to be implemented by applications to receive SSO token events.

Sample SSOTokenEvent Code. The `SSOTokenEvent` class can be used in the following way to get SSO Token events:

Code Example 5-5 Sample `SSOTokenEvent` Code

```
SSOTokenListener myListener = new AppTokenListener();
token.addSSOTokenListener(myListener);

where AppTokenListener is a class defined as follows:

public class AppTokenListener implements SSOTokenListener {
    public void ssoTokenChanged(SSOTokenEvent event) {
         try {
             SSOToken token = event.getToken();
             int type = event.getType();
             long time = event.getTime();
```

Code Example 5-5 Sample `SSOTokenEvent` Code *(Continued)*

```
SSOTokenListener myListener = new AppTokenListener();
            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());
        }
    }
}
```

SSOTokenID

The `SSOTokenID` class is used to identify the SSOToken object. Additionally, the `SSOTokenID` string contains a random number, the SSO server host, and server port. The random string in the `SSOTokenID` is unique on a given server. In the case of services written using a servlet container, the `SSOTokenID` can be communicated from one servlet to another either:

- as a cookie in a HTTP header; or
- as an implementation of the `SSOTokenListener` interface by the applications to receive the SSO token events.

SSOTokenListener

The `SSOTokenListener` interface provides a mechanism for applications that need notification when an SSO token expires. (It could expire if it reached its maximum session time, or idle time, or an administrator might have terminated the session.) Applications wishing to be notified must invoke the `addSSOTokenListener` method using the `SSOToken` interface; this method implements the `SSOTokenListener` interface. A callback object will be invoked when the SSO token expires. Using the `SSOTokenEvent` (provided through the callback), applications can determine the time, and the cause of the SSO token expiration.

NOTE Once an application registers for SSO Token events using `addSSOTokenListener`, any SSO token event will invoke the `ssoTokenChanged` method. The application can take suitable action in this method.

Sample API Code

Following are examples of code that illustrate various operations that can be performed by the SSO API.

User Authentication Sample Code

This code can be used to determine if a user is authenticated. (Additionally, the API can be used to perform a query on a token for information such as host name, IP address, or idle time).

Code Example 5-6 Code Sample To Determine If User Is Authenticated

```
try {
            ServletOutputStream out = response.getOutputStream();

            /* create the sso token from http request */
            SSOTokenManager manager =
SSOTokenManager.getInstance();
            SSOToken token = manager.createSSOToken(request);

            /* use isValid method to check if the token is valid
             * this method returns true for valid token, false non
              */
            if (manager.isValidToken(token)) {
                /* let us get all the values from the token */

                String host = token.getHostName();
                java.security.Principal principal =
token.getPrincipal();
                String authType = token.getAuthType();
                int level = token.getAuthLevel();
                InetAddress ipAddress = token.getIPAddress();
                long maxTime = token.getMaxSessionTime();
                long idleTime = token.getIdleTime();
                long maxIdleTime = token.getMaxIdleTime();
                out.println("SSOToken host name: " + host);
                out.println("SSOToken Principal name: " +
principal.getName());
                out.println("Authentication type used: " +
authType);
                out.println("IPAddress of the host: " +
                          ipAddress.getHostAddress());
            }
          /* try to validate the token again, with another method
              * if token is invalid, this method throws exception
              */
            manager.validateToken(token);

            /* get the SSOTokenID associated with the token */
            SSOTokenID tokenId = token.getTokenID();
```

Code Example 5-6 Code Sample To Determine If User Is Authenticated *(Continued)*

```
try {
            String id = tokenId.toString();

            /* print the string representation of the token */

            out.println("The token id is " + id);

            /* set properties in the token. We can get the values
             * of set properties later
             */
            token.setProperty("Company", "Sun Microsystems");
            token.setProperty("Country", "USA");
            String name = token.getProperty("Company");
            String country = token.getProperty("Country");

            out.println("Property: Company is - " + name);
            out.println("Property: Country is - " + country);

            out.println("SSO Token Validation test Succeeded");
            /* add a listener to the SSOToken. Whenever a token
             * event arrives, ssoTokenChanged method of the
             * listener will get called.
             */
            SSOTokenListener myListener = new
SampleTokenListener();

            token.addSSOTokenListener(myListener);
            out.flush();
        } catch (Exception e) {
            System.out.println("Exception Message: " +
e.getMessage());
            e.printStackTrace();
        }
    }
}
```

In some cases, it might be more efficient and convenient to use `SSOTokenManager.validateToken(token)` than `SSOTokenManager.isValidToken(token)`. `SSOTokenManager.validToken(token)` throws an exception when the token is invalid, thus terminating the method execution right away.

Get Token Sample Code

This sample code can be used to get the SSO token if the `SSOtokenID` string is passed to the application.

Code Example 5-7 Code Sample To Get Token from Token ID

```
try {
            /* create the sso token from SSO Token Id string */
           SSOTokenManager manager=SSOTokenManager.getInstance();
           SSOToken token = manager.CreateSSOToken(tokenString);
            * let us get the SSOTokenID associated with the token
*/
            SSOTokenID id = token.getTokenID();

            String tokenId = id.toString();

            /* print the string representation of the token */

            System.out.println("The token ID is " + tokenId);

            /* set properties in the token. We can get the values
             * of set properties later */

            token.setProperty("Company", "Sun Microsystems");
            token.setProperty("Country", "USA");
            String name = token.getProperty("Company");
            String country = token.getProperty("Country");

           System.out.println("Property: Company is - " + name);
            System.out.println("Property: Country is - " +
country);

            System.out.println("SSO Token Validation test
Succeeded");
            /* add a listener to the SSOToken. Whenever a token
             * event arrives, ssoTokenChanged method of the
             * listener will get called.
             */
            SSOTokenListener myListener = new
SampleTokenListener();

            token.addSSOTokenListener(myListener);
        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
           SSOTokenManager manager=SSOTokenManager.getInstance();
            SSOToken token = manager.CreateSSOToken(tokenString);
        }
    }
```

Listen For Event Code Sample

Applications can listen for SSO token events. It is possible that while a user is using an application, an SSO token may become invalid because, for example:

- the user's access times out because of the maximum time limit; or,

- the user fails to log out of an application and the idle time-out expires.

The application must be informed of these *events* to follow-up on the invalid token by terminating the user's access. The following two sample codes can be used to get token events.

Code Example 5-8 Code Sample To Register For SSOToken Events

```
SSOTokenListener myListener = new SampleTokenListener();
token.addSSOTokenListener(myListener);
```

where `SampleTokenListener` is a class defined as:

Code Example 5-9 Code Sample Defining `SampleTokenListener` Class

```
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 is: " + id.toString());

          if (SSOTokenManager.getInstance().isValidToken(token))
{
                System.out.println("Token is Valid");
            } else {
                System.out.println("Token is Invalid");
            }

            switch(type) {
            case SSOTokenEvent.SSO_TOKEN_IDLE_TIMEOUT:
                System.out.println("Token Idel Timeout event");
                break;
            case SSOTokenEvent.SSO_TOKEN_MAX_TIMEOUT:
                System.out.println("Token Max Timeout event");
                break;
            case SSOTokenEvent.SSO_TOKEN_DESTROY:
                System.out.println("Token Destroyed event");
                break;
            default:
                System.out.println("Unknown Token event");
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
```

Code Example 5-9 Code Sample Defining `SampleTokenListener` Class *(Continued)*

```
public class SampleTokenListener implements SSOTokenListener {
            }
      }
}
```

After the application registers for SSO token events using `addSSOTokenListener`, any SSO token events will invoke the `ssoTokenChanged()` method. The application can take a suitable action in this method.

Sample SSO Java Files

DSAME installs three groupings of sample Java files with instructional text. With these files, a developer can create an SSO token in several ways:

1. An SSO token can be created for an application that runs on the DSAME server.
2. An SSO token can be created for an application that runs on a server other than the DSAME server.
3. An SSO token can be created by a session ID string can be passed through the command line.

The files needed to perform these actions can be found in the *`Install_Directory`*`/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 DSAME 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 DSAME 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.

Multi-JVM Support

DSAME can run on iPlanet Application Server (AS) which supports a multi-JVM environment. In this scenario, based on load balancing, a SSO service could run in any JVM. If one JVM receives the `createSSOToken()` request, subsequent validation requests must be directed to the same JVM otherwise, the SSO service will send an invalid token response. For this purpose, DSAME uses the sticky session feature of the AS; it is turned on automatically when the DSAME is installed. For more information, see the documentation that comes with iPlanet Application Server.

NOTE	The SSO APIs cannot be used in a multi-JVM environment.

Chapter 6

Logging

The iPlanet Directory Server Access Management Edition (DSAME) provides a logging module as a means of recording information such as user activity, traffic patterns, and authorization violations. In addition, DSAME includes a Logging API so that applications can take advantage of the logging function. This chapter explains the component and the API. It contains the following sections:

- Overview
- Log Message Formats
- Logging API
- Sample Logging Code

Overview

The Logging component enables all DSAME services to record information that might be useful to an administrator. This allows tracking of who is accessing what resources in one centralized location. It accepts requests to provide logging operations which include writing messages to logs, reading logs, listing log files and deleting log files. Examples of information logged might include user access denials and approvals, traffic patterns, authorization violations and code exceptions. The component allows logs to be written to either a relational database or flat files. It contains the following modules:

- A Logging service which contains the configuration parameters for the logging function and accepts and processes logging requests.
- Java API which can be integrated into Java applications in order to allow them to access the Logging service.

Logging Architecture

An application accesses the Logging service by calling the Logging API. Upon receiving a request, the Logging service loads the configuration data stored in the LDAP DS using the DSAME SDK. (This information might include the log format, the log's maximum size and the log's location.) Any exception message will be logged, based on these configuration values. On an error, a `LoginException` is thrown by the Logging service.

NOTE The API can reside on the same server as the service or on a remote one. If the Logging interfaces are remote, the Communication Component (PLL) is used to send the request to the Logging service.

Logging Service

The Logging service holds the attributes and values for the DSAME logging function. The values are applied across the configuration and are inherited by every configured organization. The Logging Attributes are:

- Max Log Size
- Number of History Files
- Log Location
- Logging Type
- Database User Name
- Database User Password
- Database Driver Name

More information on these attributes and the Logging service can be found in the *iPlanet Directory Server Access Management Edition Administration Guide*.

Log Message Formats

DSAME supports both flat-file based logging and JDBC logging. Log records can be stored in either a flat file or in a table of a relational database. The following sections explain the formats of both record types.

File Format

The Logging service uses `DATE/TIME&&Domain&&LoginID&&Type&&DATA` to log messages. This format is explained below. Code Example 6-1 below illustrates how a log record formatted for a file would look.

- `TIME` is the date (yyyy / mm / dd) and time (hh:mm:ss) at which the log message was recorded.
- `DOMAIN` is the DSAME organization to which the user belongs.
- `LOGINID` is the ID of the user attempting to access the application.
- `TYPE` is the application writing the log.
- `DATA` is the description of the user activity, errors or other useful information which the application wants to log.

Code Example 6-1 File Formatted Log Record Sample

```
&&TIME=2002/04/25 13:24:47
PDT&&DOMAIN=o=iplanet.com&&LOGINID=uid=amAdmin,ou=People,o=iplan
et.com&&

TYPE=amConsole&&DATA=Registered service iPlanetAMAuthService
```

Database Format

For applications using a relational database to log messages, the message is stored in a database table.

NOTE There is a limitation in the log name length for Oracle JDBC logging: the length of the log name cannot exceed 30 characters. Oracle does not support names longer than 30 characters.

The database schema is as follows:

Code Example 6-2 Database Message Format

Column Name	Data Type	Description
TIME	VARCHAR(200)	Date (yyyy/mm/dd hh:mm:ss)
DOMAIN	VARCHAR(100)	User's DSAME Organization
LOGINID	VARCHAR(50)	Login User's ID.
TYPE	VARACHAR(20)	Application type.
DATA	VARCHAR(300)	Message to be logged.

Logging API

The Logging API provides log management tools for DSAME services as well as providing a set of Java classes for applications to create, retrieve, submit, or delete log information. The API can be used, for instance, to develop log auditing capabilities. The main classes are `LogManager` and `LogRecord`. They are contained in the package `com.iplanet.log`.

NOTE The Overview page for the complete set of public Javadocs can be accessed at *`Install_Directory`*`/SUNWam/docs/index.html`.

LogManager Class

This `LogManager` class provides the methods for applications to use in creating, retrieving, submitting, and deleting log information. It also provides a method to access a list of log names that have been created by all the applications. This class provides methods and must be instantiated in order to use the `LogRecord` class.

- The `Create()` method creates a log in the designated logging location.
- The `Write()` method records a single piece of log information or a `LogRecord`. It allows an application to submit a logging message to a predetermined log.

LogRecord Class

The class `LogRecord` class provides the means to represent the information that needs to be logged. Each instantiation represents a single piece of log information or `LogRecord`. This information comes from the application. This class provides two methods and must be instantiated in order to use the `LogRecord` class.

- The `getRecType()` method retrieves the log type or key of a single log record.
- The `getRecMsg()` method retrieves the log data or value of a single log record.

Logging Exceptions

There are a number of exceptions that can be thrown using the Logging APIs. The generic `LogException` is probably the most common. It signals an error condition while logging a message. Other exceptions include:

- `ConnectionException`—This exception is thrown when the connection to the database fails.
- `DriverLoadException`—This exception is thrown when the JDBC driver load fails.
- `InvalidLogNameException`—This exception is thrown when the log name is invalid.
- `LogAlreadyExistException`—This exception is thrown when the log already exists.
- `LogCreateException`—This exception is thrown when log creation fails.
- `LogDeleteException`—This exception is thrown when the log deletion fails.
- `LogException`—A `LogException` is thrown when applications are denied log access because they don’t have the privileges or a valid session.
- `LogFatalException`—This exception is thrown when a fatal error occurs.
- `LogHandlerException`—A `LogException` is thrown when a log handler error is encountered.
- `LogInactiveException`—A `LogException` is thrown when the log is in inactive status. (Inactive / active status is not currently supported.)
- `LogInvalidSessionException`—This exception is thrown when an application accesses a log which does not exist.

- `LogNotFoundException`—This exception is thrown when an application accesses a log which does not exist.
- `LogPrivDeniedException`—A `LogException` is thrown when the access privilege is denied.
- `LogProfileException`—A `LogException` is thrown when access privilege is denied.
- `LogReadExceedsMaxException`—A `LogException` is thrown when the log size exceeds the maximum size defined in the Logging service.
- `LogReadException`—A `LogException` is thrown when an error is encountered in retrieving the log information.
- `LogTypeException`—This exception is thrown when a log type error occurs.
- `LogWriteException`—This exception is thrown when the log record submission fails.
- `NullLocationException`—This exception is thrown when the location is null.

Sample Logging Code

Code Example 6-3 below provides sample code that shows how to use the DSAME logging classes discussed above.

Code Example 6-3 Logging API Sample

```
LogManager lm = new LogManager(SampleSSOSession);

try {
    lm.create("SampleLog");
    }
} catch (Exception e) {
    System.out.println("Error: " + e.getMessage();
}

try {
    LogRecord lr = new LogRecord("SampleType", "SampleData");
    log.write(lr, "SampleLog");
} catch(Exception e) {
    System.out.println("Error: " + e.getMessage());
}
```

Recorded Events

By default, DSAME currently logs events in three logs:

SSO-related Logs

The Logging component logs the following events for the SSO component:

- `Login`
- `Logout`
- `Session Idle TimeOut`
- `Session Max TimeOut`
- `Failed To Login`
- `Session Reactivation`
- `Session Destroy`

The log is called `amSSO`.

Console-related Logs

The Logging component records the creation, deletion and modification of identity-related objects, policies and service including, among others, Organization, Organizational Unit, User, Role, Policy and Group. It also records modification of all user attributes including password and the addition or removal of users to or from roles and groups, respectively. The log is called `am Console`.

NOTE The Web Agents are responsible for logging exceptions related to resource access or denial; in other words, policy. For more information, see the Web Agent documentation.

Authentication-related Logs

The Logging component logs the events for the Authentication component. The log is called `am Authentication`.

Chapter 7

Utility APIs

The iPlanet Directory Server Access Management Edition (DSAME) provides utility application programming interfaces (APIs) that can be used by applications. This chapter explains these APIs. It contains the following sections:

- Overview
- API Summary

Overview

The utilities package is called `com.iplanet.am.util`. It contains utility programs that can be used by applications accessing DSAME. The APIs include:

- `StatsListener`
- `AdminUtils`
- `AMClientDetector`
- `Debug`
- `Locale`
- `Stats`
- `SystemProperties`
- `ThreadPool`

API Summary

Following is a summary of the utility APIs and their functions.

StatsListener

The `StatsListener` interface must be implemented by each module in order to print the statistics. This interface invokes the `printStats()` method.

AdminUtils

This class contains the methods used to retrieve TopLevelAdmin information. The information comes from the server configuration file (`serverconfig.xml`).

AMClientDetector

This is a utility that gets the client type. It executes the Client Detection Class (provided in Client Detection service) to get the client type. The default client type will be returned if there is no Client Detection Implementation provided.

Debug

Debug allows an interface to file debug and exception information in a uniform format. It supports different levels of filing debug 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 may hurt performance when abused. 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 to maximize application performance.

Locale

This class `Locale.java` is a utility that provides the functionality for applications and services to internationalize their messages.

Stats

This class writes statistics information in a uniform format. It supports different states of filing information:

- OFF statistics is turned off.
- FILE statistics information is written to a file.
- CONSOLE statistics information is written to the console.

The Stats service uses the property file, `AMConfig.properties`, to set the default stats level and the output directory where the stats files will be placed. The properties file is located (using ResourceBundle semantics) from one of the directories in the CLASSPATH.

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 brand 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.

Chapter 8

iPlanet Directory Server And DSAME

iPlanet Directory Server Access Management Edition (DSAME) uses iPlanet Directory Server (DS) to store its data. Certain features of the LDAP-based DS are also used by DSAME to help manage the data. This chapter contains information on these DS features and how they are used. It contains the following sections:

- Overview
- Roles
- Access Control Instructions (ACIs)
- Class Of Service

Overview

DSAME has been built to work with DS. They are complementary in architecture and design data. DSAME needs an underlying directory server to function. Use of the directory, though, is not exclusive to DSAME and, therefore, needs to be treated as a completely separate deployment. For more information on the directory server, see the iPlanet Directory Server documentation.

Roles

Roles are a DS entry grouping 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 role itself is defined in an LDAP entry as a role object and is identified by the DN of the object. DS has a number of different types of roles but DSAME can only manage one of them: the managed role.

NOTE The other DS role types can still be used in a directory deployment; they just can not be managed by DSAME.

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. Then, when new contractors start, the administrator would assign them this role instead of setting separate attributes in the contractor entry. If the contractor were then to become a full-time employee, the administrator could just re-assign them a different role.

Managed Roles

With a managed role, role membership is defined in each role's member entry and not in the role's definition entry. An attribute which designates membership is placed upon each entry which belongs to the role. This is in 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 DSAME roles can only be configured directly under organization or sub-organization entries.

Role Definition Entry

A role definition entry is a LDAP entry where the role's characteristics are defined. Below is a sample of a manager role definition entry.

Code Example 8-1 LDAP Role 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.

Role Member Entry

A role member entry is an LDAP entry in which the role is applied. The `nsRoleDN` attribute indicates that the entry is a member of a managed role identified by the DN of its role definition entry; in Code Example 8-2 below, the DN identifies Code Example 8-1 on page 135 as the role definition entry `cn=managerrole,dc=siroe,dc=com`.

Virtual Attribute

When a role member entry that contains the `nsRoleDN` attribute is returned by DS, `nsRoleDN` will be aliased to the `nsRole` attribute on that same entry. `nsRole` will carry a value of any managed, filtered or nested roles assigned to the user (such as `ContainerDefaultTemplateRole`). The LDIF Code Example 8-2 on page 136 includes this virtual attribute when returned by DS only.

Code Example 8-2 LDAP Role 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 DSAME Uses Roles

DSAME uses roles to apply access control instructions. When installed, the DSAME application configures access control instructions (ACIs) to define administrator permissions. These ACIs 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 access permissions.

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 DSAME 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 DSAME attempts to process two templates that are set to the same priority level, DS arbitrarily picks one of the templates to return. For more information, see the iPlanet 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 also 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. DSAME 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, DSAME 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. DSAME 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 tree is displayed as a starting point. For more information on the `iplanet-am-role-type` attribute, see "Role Creation," on page 136.

NOTE If the attribute has no value, a search from DSAME root is performed for all container-type objects; the highest object in the 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 DITs with many container entries.

Access Control Instructions (ACIs)

NOTE This section refers to ACIs as they are applied to administrative roles only. There are other ACIs which are created and used in DSAME but do not apply to this topic or to roles.

Access control in DSAME is implemented using DS roles. Users inherit access permissions based on their role membership and parent organization. DSAME installs pre-configured administrator roles that define access permissions for administrators that are dynamically created when a group, organization, container or people container is configured. (They are `Organization Admin`, `Organization Help Desk Admin`, `Group Admin`, `Container Admin`, `Container Help Desk Admin` and `People Container Admin`.) These roles apply a set of default access control instructions (ACIs) that define read and write access to the entries in the corresponding object. For example, when an organization is created, the DSAME SDK creates an `Organization Admin` role and an `Organization Help Desk Admin` role. The permissions are read and write access to all organization entries and read access to all organization entries, respectively.

NOTE The DSAME 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 DSAME console Administration XML service file, `amAdminConsole.xml`. This file contains two global attributes that define ACIs for use in DSAME: `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 DSAME 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 139.

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 set using DSAME for use in administrator-type roles follow a different format than those set using the DS. The format of the predefined DSAME 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 these ACIs allow.
- `DN:ACI`—There can be any number of `DN:ACI` couplets separated by the `##` symbols. The SDK will get and set each couplet in the `DN` entry. 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 DIT 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 140 for examples of ACI formats. Additionally, more complete ACI information can be found in the iPlanet Directory Server documentation.

NOTE If there are duplicate ACIs within the default permissions, the SDK will print a debug message.

Default ACIs

Following are the default ACIs installed by DSAME. They are copied from a DSAME configuration whose top-level organization is 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 DSAME 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 DS that allows attributes to be created and managed in a single central location, and dynamically added to user entries. Attribute values are not stored with 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 entries:

- CoS Definition Entry—This entry identifies the type of CoS being used (ClassicCoS). It contains all the information, save 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 shared attribute values. Changes to the attribute values are automatically applied to all entries within the scope of the CoS.

The CoS definition entry and template entry interact to provide attribute information to their target entries, which is any entry within the scope of the CoS. Only those services which have dynamic or policy attributes use the DS CoS feature; no other services do.

NOTE For additional information on the CoS feature, see the iPlanet 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. 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 could be assigned to a role or organization, thus creating possible conflicts. When this happens, DSAME 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 DSAME 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 41 of Chapter 2, "DSAME And XML" or the iPlanet Directory Server documentation.

NOTE	Conflict resolution is decided by the DS before the entry is returned to DSAME. DSAME allows only the definition of the priority level and CoS type.

Existing Applications

If a customer is using an existing application and wants to manage its attributes using the DSAME console, a LDAP schema is probably defined and has been loaded into the DS. If DS does not already have the existing application's attributes and object classes loaded, then it needs to be updated using the DS console or the `ldapmodify` command line interface. The schema update needs to be completed before loading the application's created XML service file. Other options for adding or modifying DS schema can be found in the iPlanet Directory Server documentation or in the *iPlanet Directory Server Access Management Edition Installation and Configuration Guide*.

Index

A

B

L

M

O

P

R

S

T

U

V

W

X