Programmer's Guide

iPlanet™ Directory Server
Access Management Edition

Version 5.0

December 2001

Contents

List of Code Examples

About this Guide

This Programmer's Guide provides information on how to customize the public interfaces in *iPlanet Directory Server Access Management Edition*.

This preface contains the following sections:

- Who Should Use This Book
- What You Are Expected to Know
- How This Book Is Organized
- The iPlanet Directory Server Access Management Edition Documentation Set
- Documentation Conventions Used in This Manual
- Related Information

Who Should Use This Book

This document is intended for service developers or programmers who want to create customized services by using the public interfaces in the *iPlanet Directory Server Access Management Edition*(DSAME) 5.0 product. (The term "public interfaces" or "exposed interfaces" throughout this book refers to any APIs, SPIs, XML over HTTP, or HTML over HTTP interfaces that can be used to create custom services or applications.)

What You Are Expected to Know

This book is intended for use by service developers or programmers who want to create custom services or applications to work with iPlanet™ Directory Server Access Management Edition. It's essential that you understand directory technologies and have some experience with Java and XML programming

languages in order to customize the Single Sign-On APIs to create a custom SSO solution, the Pluggable Authentication Service Provider Interfaces (SPIs) to create a custom pluggable authentication service, or to import custom service XMLs or batch update XML files into DSAME.

You will get the most out of this guide if you are familiar with directory servers and Lightweight Directory Access Protocol (LDAP). Particularly, you should be familiar with iPlanet Directory Server and the documentation provided with that product.

Refer to the *Directory Server Access Management Edition Installation and Configuration Guide* for conceptual information on DSAME and how it can be used in your network, and for information on installing and configuring DSAME in your networked environment. Also refer to the *Directory Server Access Management Edition Administration Guide* for information on how to manage and customize services in your network.

How This Book Is Organized

This book contains the following chapters and appendices:

Overview of the Programmatic Interfaces in DSAME provides an introduction to some of the tools available.

Pluggable Authentication SPI describes requirements for writing a supplemental authentication module, and provides information about customizing the authentication pages.

HTML Templates provides information for a service developer or programmer to change the authentication (login, logout, and timeout) pages for different organizations.

Single Sign-On provides information on how to customize the public Single Sign-On APIs to create a custom Single Sign-On solution.

Understanding XMLs and DTDs provides information on DSAME service XMLs, and attributes and elements supported by the Services Management Services DTD (`sms.dtd`); and information on the batch update XML files and the attributes and elements supported by the batch update XML DTD (`amAdmin.dtd`).

Using the Command Line Interface describes the command-line interface (`amadmin`), how to write and import custom service XML files (such as a custom pluggable authentication service or module) into DSAME, and how to write and import batch XML files to update various user objects (such as organizations, users, roles, people containers, groups, etc.) in the Directory Information Tree (DIT).

The iPlanet Directory Server Access Management Edition Documentation Set

The DSAME documentation set contains the following titles:

- *Installation and Configuration Guide* describes DSAME and provides details on how to plan and install DSAME on Solaris systems.
- *Administration Guide* documents how to manage user and service data in an iPlanet Directory Server Access Management Edition system once it has been installed.
- *Programmer's Guide* documents how to customize DSAME interfaces.
- 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.
- Online help for users and online help for system administrators
  - `http://`*`yourserver:port`*`/docs/en_US/javadocs`

  In the installed version of the product, the Javadocs can be found in:
  - *`DSAME_root`*`/web-apps/services/docs/en_US/javadocs`

  It is recommended that you also refer to the iPlanet Directory Server 5.1 documentation for information on iPlanet Directory Server:
  - `http://`*`yourserver:port`*`/docs/en_US/javadocs`

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.

`http://docs.iplanet.com`

Documentation Conventions Used in This Manual

In the iPlanet Directory Server Access Management Edition documentation (such as this guide) 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 that are installed and running on a dedicated Web Server. The dedicated Web Server is installed for you automatically when you install the Police and Management Services.
- *Web Server that runs DSAME* refers to the dedicated Web Server where the Policy and Management services are 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

DSAME Programmer's Overview

The *Directory Server Access Management Edition Programmer's Guide* describes how service developers and programmers can customize the following public interfaces in DSAME 5.0:

- the Java client Application Programming Interfaces (APIs) that enable service developers to customize the Single Sign-On solution;
- the Java Pluggable Authentication Service Provider Interfaces (SPIs) which allow service developers to create a custom pluggable authentication service;
- the client Authentication Service interfaces which allow service developers to customize the authentication login, logout, and timeout screens by modifying the HTML templates;
- the command line interface (`amadmin` tool) which enables service developers and customization engineers to import custom service XML files, and batch update XML files to perform operations such as creates, deletes, and gets on various directory objects in the DIT (such as users, groups, roles, people containers, etc.).

This chapter contains the following sections:

- Extending iPlanet Directory Server Access Management Edition
- Which APIs, SPIs, and XML over HTTP Interfaces you can customize
- Overview of Service Development Process when extending DSAME Interfaces

The Programmer's Guide provides information for programmers customizing iPlanet Directory Server Access Management Edition (DSAME) software. It documents the public Java application programming interfaces (APIs) and service provider interfaces (SPIs) that are included in the iPlanet Directory Server Access

Management Edition product, as well as documents the exposed HTTP over XML interfaces. For example, you can use the Single Sign-On APIs and the Authentication SPIs to integrate the application with the iPlanet Directory Server Access Management Edition software and use single sign-on capabilities.

The Pluggable Authentication SPIs and Single Sign-On SPIs are for authenticating users and issuing a single-sign-on (SSO) token. The SSO APIs provides Java interfaces that validate the SSO tokens and maintain authentication credentials for the user. The SSO APIs can be used to provide a mechanism by which users need to authenticate only once, and then can access multiple we-based applications without having to re-authenticate. Additionally, it provides interfaces for applications to store generic key-value pairs and to register callback listeners, which will be invoked when the SSO token is destroyed.

DSAME's single-sign-on solution is primarily intended for web-based applications; however, service developers can use the Single Sign-On APIs to write Java-based services.

The Single Sign-On APIs provide methods to:

- get the SSO token from its string representation (token ID)
- validate or invalidate the SSO token
- get the principal name of the authenticated user
- get the token ID
- get the authentication type
- get the authentication level
- get the hostname of the client that sent the SSO token
- method to get the value of a property stored in the SSO token
- add a listener that will be called when the token is destroyed, or has reached its maximum idle timeout, or has reached its maximum session timeout.
- get the SSO token in a string format
- check if two SSO tokens are equal.
- get the SSO token from its string representation that is provided by the token ID.
- notify applications when the SSO token expires.
- get the time when the token expired.
- get the cause for the token to expire.

The third major interface that can be customized by programmers is the HTML templates. Service developers or programmers can modify the HTML templates to create custom authentication login, logout, and timeout pages for different organizations in DSAME console.

The fourth customizable interface in DSAME 5.0 is the XML interface. Service developers can use the `amadmin` CLI tool to import custom service XML files such as when creating a custom pluggable authentication service. Also, they can use the `amadmin` tool to import batch update XML files that update objects in the DIT (such as creating, reading, and deleting roles, users, organizations, groups, people containers, and services). Developers writing programs that will communicate with the exposed XML over HTTP interface to iPlanet Directory Server Access Management Edition need to understand and be able to use eXtensible Markup Language (XML) and HTTP.

NOTE Detailed information on the public APIs and SPIs in DSAME 5.0 (Single Sign-On and Authentication) is available in the Javadocs. (The Javadocs are located in `<dsame_root>/web-apps/services/docs/en_USjavadocs`.

Extending iPlanet Directory Server Access Management Edition

DSAME can be extended in several ways, which is described in this section. If additional authentication capabilities are needed, use the public Java Pluggable Authentication SPIs to create them. To add Java-based services that can make use of the single sign-on solution, use the Single Sign-On APIs and the Authentication SPIs to integrate them into the iPlanet Directory Server Access Management Edition framework. (Note that the Single Sign-On solution is primarily a web-based solution, but can be extended using the Java APIs.)

DSAME can be extended by adding new or custom services or applications, such as authentication plugin modules or custom single sign-on solutions. Typically, for most services, DSAME and Directory Server must be updated with any new objectclasses and attributes that an application or service will use, in order for DSAME to manage those service attributes. (For general information on the necessity of updating the schema in Directory server when adding a service, see Chapter 5, “Understanding DSAME XMLs and DTDs.” For information on updating schema when adding a custom authentication module, or a single sign-on solution, see Chapter 2, “Pluggable Authentication SPI” and Chapter 6, “Using the Command Line Interface,” respectively.)

Which APIs, SPIs, and XML Interfaces You Can Customize

This section gives brief descriptions of which interfaces are customizable by programmers in the DSAME 5.0 release:

- Authentication SPI — Java interfaces for creating custom pluggable authentication services. You can use the exposed Java Service Provider Interfaces (SPIs) and classes to create a server-side plug-in authentication module.
- Single Sign-On Client APIs — Single Sign-On component provides Java interfaces so that applications can participate in the SSO solution. These APIs are intended primarily for web-based applications, but they can be extended to any Java-based applications.
- `amadmin` CLI tool — The amadmin CLI tool is considered a declarative interface. The two primary purposes of the amadmin command line interface tool are to import service schema and configuration data into DSAME, and to perform batch updates to the Directory Server (such as creating, getting, and deleting roles, users, organizations, groups, people containers, and services).

The iPlanet Directory Server Access Management Edition APIs, SPIs, and XML over HTTP interfaces that are customizable in the DSAME 5.0 release fall into several broad categories:

To do:	Use these APIs/SPIs/XML over HTTP Interfaces
Single Sign-On	Single Sign-On APIs are programmatically customizable. Customers can use the seven public single sign-on APIs to create custom services.
Authentication Extension	Three Authentication service provider interfaces (SPIs) are public in this release so customers can write their own supplemental authentication module (SPI—service provider interface) to plug into iPlanet Directory Server Access Management Edition.

To do:	Use these APIs/SPIs/XML over HTTP Interfaces
Service XML and Bulk Updates XML Files	Service developers and/or customization engineers can extend the service XMLs to create their own custom services or applications to integrate into DSAME. Additionally, they can use the bulk operations XMLs to create, get, and delete user objects in the DIT (Directory Information Tree) such as roles, users, organizations, groups, people containers, and services.

Overview of Service Development Process When Extending DSAME Interfaces

This section describes the development process (at a high level) when extending the public DSAME APIs, SPIs, and/or XML interfaces to create a custom service to integrate into the iPlanet Directory Server Access Management Edition product.

The following main steps outline the process:

1. Define high-level application requirements.
2. Determine which iPlanet Directory Server Access Management Edition APIs, SPIs, or service XMLs (Authentication, etc.) support the high-level requirements.
3. Define the iPlanet Directory Server Access Management Edition attributes.
4. Define the privileges that determine the policy for the service. Create an XML file to define the service attributes as they will display, and thus be manageable from, the DSAME console.
5. Import the XML file to the DSAME server with the `amadmin` CLI tool.
6. Configure and modify the services through the DSAME console.

For more detailed information on creating services to integrate into DSAME, see the section “High Level Flow for Creating and Registering Services,” on page 157 of this guide.

Each XML DTD specifies the content and format of the information that can be sent to and received from the iPlanet Directory Server Access Management Edition services. (See Chapter 5, “Understanding DSAME XMLs and DTDs” and Chapter 6, “Using the Command Line Interface” for information on the service XML and batch update XML files used in DSAME, and how to use the command line interface to import the service XMLs and the batch updates XML files into DSAME.)

Chapter 2

Pluggable Authentication SPI

DSAME provides an Authentication and Single Sign-On (SSO) framework with a plug-in architecture. The authentication mechanisms supported in DSAME 5.0 are:

- LDAP—Authenticates a user using LDAP (Lightweight Directory Access Protocol).
- Certification-based—Authenticates a user using a Personal Digital Certificate (PDC).
- RADIUS—Authenticates a user through an external RADIUS server.
- Membership (Self-registration)—For new user self-registration and login.
- Anonymous—Any user can log into DSAME as an anonymous user with limited access. Anyone can access it without providing a bind DN or password.

The plug-in framework for authentication provides a way for customers to plug in their own authentication mechanisms. The SSO solution provided by DSAME is primarily for web-based applications. This solution provides a mechanism by which end users need to authenticate only once, and then can access multiple web-based applications without having to re-authenticate. (See “Authentication Process Overview” on page 29 for information on authentication architecture and how authentication interacts with the SSO feature.)

Overview of Pluggable Authentication SPI

This chapter describes requirements for writing a supplemental authentication module (SPI—service provider interface) to plug into iPlanet DSAME and provides information on customizing the authentication pages.

The section “Authentication Process Overview” on page 29 describes the architecture of the authentication component and its public interfaces. The Section “Writing and Integrating a Pluggable Authentication Module” on page 42 gives information on how to write and integrate a pluggable authentication service (module). See Table 2-1 on page 28 for a list of common tasks related to customizing pluggable authentication modules, and where to find information on them.

Table 2-1 Tasks to Customize Authentication

If you want to	Do
Change login prompts	Edit the `.properties` file associated with the login screen that you want to change. See “Understanding the screen.properties File,” on page 36.
Add authentication capability	• Create a “screen”`.properties` file for the authentication module you write and intend to integrate into DSAME. See “Understanding the screen.properties File,” on page 36. • Write and integrate an auth module.“Writing and Integrating a Pluggable Authentication Module,” on page 42.
Selectively enable or disable auth modules	Refer to *DSAME Administration Guide*
Customize prompts and appearance for specific organizations or sub-organizations	Refer to *DSAME Administration Guide*.

TIP Before beginning to write authentication modules, contact the iPlanet Solutions sales representative to find out if the module needed has already been written and is available from internal resources.

For additional authentication capabilities, use the information in this chapter to write a custom authentication module and integrate it into the iPlanet Directory Server Management Access Edition product.

Authentication Process Overview

The architecture of DSAME's authentication is shown in Figure 2-1 on page 30. It has three layers:

- client interface (Authentication Service) using HTML / HTTP
- framework
- authentication module interfaces (plug-in SPIs)

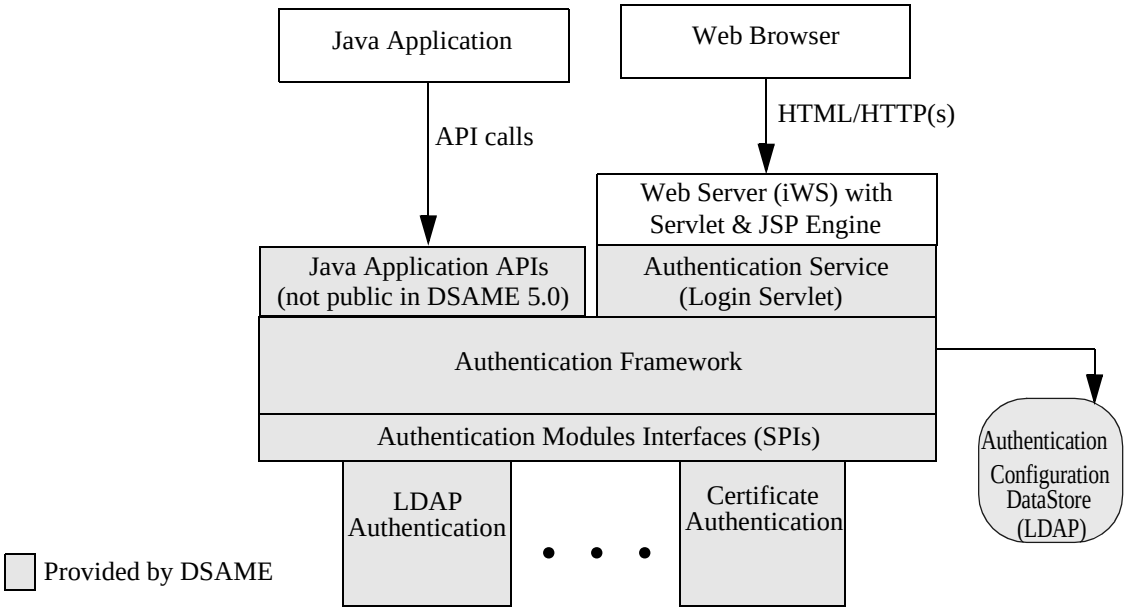
Each of these layers is discussed in the following paragraphs.

Client Interface (Authentication Service using HTML/HTTP)

DSAME authentication provides two client interfaces, only one of which is exposed in DSAME 5.0—the Authentication Service which provides an HTML over HTTP interface. (A "public" or "exposed" interface is one that customers can customize to create their own service or module.) Application developers can customize the HTML templates to create new authentication login, logout, and timeout screens. (For more information, see Chapter 3, "HTML Templates".)

Although there are two client interfaces, both use the same underlying authentication framework and authentication modules. Figure 2-1 on page 30 shows the different authentication components and how they interact within the authentication architecture.

Figure 2-1 Authentication Architecture



Authentication Service Provided as HTML over HTTP(s) Interface

The Authentication Service is provided as a service within a servlet container using the Java Servlet SDK. Thus, the authentication service can be deployed in an iPlanet Web Server or an iPlanet Application Server that supports a servlet container. As shown in Code Example 2-7 on page 54, the client interface provided by the Authentication Service is HTML over HTTP(s), which makes it convenient to use with a Web browser.

In a typical scenario, the Authentication service (which is a URL) would be a login page for an organization or a service, or users would be re-directed to the authentication service URL when they access a resource that is protected. The authentication service would then guide the user through a series of one or more screens in the process of gathering credentials (such as user name, password, employee number, etc.), based on the requirements of the authentication modules that were configured. For simple authentication modules like LDAP, the required credentials would be user name and password and could be obtained in one screen. However, for complicated challenge-response type authentication algorithms, more login screens might be required. (For information on customizing the login, logout, and timeout screens, see Chapter 2, “Pluggable Authentication SPI".)

After the user has provided the required credentials, the authentication service relies on the framework to determine if the user has been successfully authenticated. If the authentication process is successful, the user is re-directed to an organization or service home page (URL). If the authentication process fails, the user is re-directed to an error page (URL). Both the re-direction URLs can be configured by the administrator. (For information, see the *iPlanet Directory Server Administrator's Guide*.)

After a user has been authenticated successfully, he or she is issued an encrypted SSO token identity using the cookie. This SSO token can be used to access different applications without having to re-authenticate. (The SSO token identity can be thought of as similar to a session identity). Also associated with the SSO token identity are attribute-value pairs, which would have information about the user's name, authentication level, authentications performed, etc. The SSO token identity can also be configured by an administrator to expire based on timeouts. (For information, see the *iPlanet Directory Server Administrator's Guide*.)

Authentication Service Features

The features provided by the Authentication Service (HTML over HTTPs) are:

- presentation using HTML and can be accessed through any Web browser
- integration with SSO by setting the SSO cookie
- configurable support to re-direct users with successful authentication
- configurable support to handle authentication failures
- support for authentication timeout, if users do not respond with credentials within specified time.

Authentication Framework

This is the middle tier of DSAME's authentication component which connects the client interfaces to the authentication modules. Features provided by authentication framework are:

- configuration of authentication modules based on organization, role, or user
- chaining of authentication modules

The authentication component uses the Service Management Services DTD (sms.dtd) to represent its configuration in a XML file. (See Chapter 5, "Understanding DSAME XMLs and DTDs" for information on service XMLs.)

Authentication Plug-In Module Interfaces (SPI)

The authentication component provides a pluggable authentication framework allowing customers to plug in their authentication mechanism.

The authentication mechanisms supported in DSAME 5.0 are:

- LDAP
- Certificate (PKI)
- RADIUS
- Membership (self-registration)
- Anonymous
- Core—Core authentication provides configuration for the authentication service itself. This Core authentication configuration is used by all other authentication modules.

  All authentication services require that the Core authentication service be configured; it must always be configured in Admin Console. The Core authentication service is a key internal DSAME service, like Session, Logging, etc. and has its own profile. The Core authentication service can be considered the “driver” of the other authentication services. The Core authentication can call any of the other configured authentication services (LDAP, RADIUS, etc.)

  The Core authentication service lets you select which authentication modules users will see in Admin Console when they go to an organization, and lets you select when a user gets redirected to another login page, which page (URL), etc. The other authentication services do things like validate credentials.

Overview of Integrating Authentication Modules in DSAME

Every authentication service in DSAME has the following components it needs and interacts with:

- a screen `.properties` configuration file; for example, `LDAP.properties` or `AuthenticationSample.properties`;

  The authentication framework loads these properties into the display, and passes your input to the authentication module. It displays each screen one at a time, as part of the authentication process. Your customized authentication plug-in module can similarly read and display input and display it on the screen as part of the authentication process. The HTML for the authentication states is dynamically generated based on the parameters set in the `.properties` configuration file for the authentication module developed.

NOTE The HTML that is dynamically generated can be overridden. See the "Understanding the screen.properties File," on page 36 for more information.

- an I18N .properties file, sometimes also called a "resource bundle" .properties file; for example, `LDAP.properties`. This file holds all the key / value pairs for the I18N key defined in the Java and XML files.
- one factory class that implements `com.iplanet.authentication.spi.AuthenticationModuleFactory`
- one module class that extends `com.iplanet.authentication.spi.AuthenticationModule`
- the `amAuth.xml` file, which is used by all the internal authentication modules, and an XML file for your customized authentication module or service (for example, `<your_custom_auth_module>.xml` file).

After the authentication module has been written and integrated, an administrator must configure it from Admin Console so that the module knows where to retrieve this information. For information on configuring authentication modules, see the *iPlanet Directory Server Access Management Edition Installation Guide.*

Where to find the Public Javadocs for Authentication SPIs

The Javadocs for the following three public authentication interfaces

- com.iplanet.authentication.spi.AuthenticationModuleFactory
- com.iplanet.authentication.spi.AuthenticationModule
- com.iplanet.authentication.spi.AuthenticationException

can be found in the `javadocs` directory of the product build, or at

`http://`*`yourserver:port`*`/docs/en_US/javadocs`

Classes and Methods you must Implement when creating a custom Authentication Service

The following sections provide information on the classes and methods you must implement in the sample authentication module, or when writing your own custom authentication service.

AuthenticationModuleFactory Methods that must be Implemented

The DSAME authentication framework implements the factory model for the pluggable authentication modules. You must implement the

- `com.iplanet.authentication.spi.AuthenticationModuleFactory` class.
- newAuthenticationModule() method—This method will create a new instance for the pluggable module you implemented.

AuthenticationModule Methods that must be Implemented

You must implement the following three methods when writing a new pluggable authentication module:

- The `init()` method which reads the hostname, password, etc.
- The second method you need to override is `validate()`. This is how you control the flow of screen. The validate method validates the user. This method is called by the authentication framework.
- The third method you must override is `getUserTokenId()`. It gets the user token ID. The `getUserTokenId()` method tells the authentication framework who the user (who has just authenticated) is.

The authentication framework controls the entire login process and which authentication module(s) are used to authenticate users.

Do You need to create a Service XML for your custom Authentication Service

Writing a service XML file is optional, whether it is for a service other than authentication, or for a custom authentication service. The only case where service developers need to write a service XML file (such as for a custom authentication service) is when he/she wants certain configuration parameters (attributes) to be manageable from the DSAME Console. The primary purpose of the service XML files is for specifying any attributes (configuration parameters) that service

developers want users and administrators to be able to configure from the DSAME Admin Console. The primary purpose of the service XML files is to feed in configuration parameters with default values set, to make them configurable from Admin Console. The APIs handle everything else behind the scenes.

It is important to note that in DSAME 5.0, because much of the DSAME SDK is not public and customizable, you cannot manage the global and organization attribute / value pairs through your external application (you cannot read and write them). The global and organization attributes are not public in this release, relative to writing customized external services.

However, when defining custom authentication service XMLs, you would typically only use the global and organization attribute types, in the DSAME 5.0 release.

Determining when an authentication service XML is necessary

To determine whether you even need to write a service XML file for your authentication service, a service developer must determine whether your authentication service has custom attributes, whether it has configuration parameters that should be configurable from the DSAME Admin Console, such as port_number, server names, etc. And if it does, does he / she want that data to be manageable and retrievable through the DSAME Admin Console. For example, do you want administrators and users to be able to go to the Authentication menu, and configure their server per organization, or port per organization, or is it a configuration parameter (attribute) that does not need to be configurable through Admin Console.

If it is determined that the configuration parameters do not need to be manageable (settable and gettable) through the DSAME Admin Console, then the service developer only needs to write a custom authentication service, but he / she would not need to create a service XML file for the authentication module.

The only reason for authentication service XMLs is to make various configuration parameters configurable from the DSAME Admin Console. (This does not necessarily apply to all service XMLs, because you can group some attributes and put them into a service to make them manageable by administrators and users through DSAME Admin Console.)

The service XML file is a way to let service developers define attributes (configuration parameters) to be configurable from Admin Console, upon which administrators and users can set and get the values. It is up to the application or service to interpret the behavior of the attribute / value pairs specified in the service XML files.

When writing a custom authentication service, only global and organization attributes are used

When defining a service XML file for a custom authentication service, you typically only use “global” and “organization” attributes in DSAME 5.0. Global and organization attribute/value pairs can be read and gotten through the pluggable authentication APIs, which has methods to read and get their attribute values. Authentication service is configured only on a per-organization basis.

When writing custom authentication service, you don’t need to update Directory server schema

Because you are defining attributes of type “global” or “organization”, you do not need to update the Directory server schema. The global and organization attributes are stored in Directory server as static blobs of data in Directory server; they are not read or stored as LDAP entries.

For the DSAME 5.0 release, because the global and organization attributes are not public in this release, service developers don’t have to update the LDAP schema in Directory server if they want to run a custom authentication service. This is because the authentication service uses an exposed pluggable Authentication API that can read and get your authentication service’s configuration.

Core Authentication Service Defines Configuration for all Authentication Services

The `amAuth.xml` file is used by the Core authentication service for its configuration. It can be thought of as a “general” or “parent” authentication service XML file for the all of the authentication services (whether internal to DSAME or external authentication applications). This file must be in the `<dsame_root>/web-apps/services/WEB-INF/config/xml` directory for the authentication modules to work. Each authentication service must also have its own service XML file, for example, `amAuthLDAP.xml`, `amAuthRADIUS.xml`, etc. Similarly, when writing a custom pluggable authentication service, you must create a new service XML file, for example, `<your_custom_auth_module>.xml` (in addition to making modifications to the Core authentication service).

Understanding the screen.properties File

The screen `.properties` file is the configuration file for an authentication module. Each authentication module must have a screen `.properties` configuration file. The file specifies the text, tokens, and password prompts for the login pages associated with the authentication module.

The HTML for the authentication states is dynamically generated based on the parameters set in the `.properties` configuration file for the authentication module developed. The HTML that is dynamically generated can be overridden with an HTML tag in the .properties file. For example, if a screen in a .properties file has the tag “HTML”, the authentication service will read in the entire HTML file specified and display that unmodified instead of dynamically generating the HTML. For example, the screen in Code Example 2-1 on page 37 wold display “mycustomhtml.html” instead of generating the HTML.

Code Example 2-1 Sample Screen calling an HTML file

```
SCREEN
TIMEOUT 120
TEXT LDAP Authentication
TOKEN Enter UserId
PASSWORD Enter Password
HTML mycustomhtml.html
```

The authentication module’s `.properties` file name must be named using the base class name of the authentication module (no package name) and the extension `.properties`. There is a sample `.properties` configuration provided in the `<dsame_root>/SUNWam/samples/authentication/providers` directory called `AuthenticationSample.properties`. There must be a .properties configuration file with the name of the class (no package name) and the extension .properties; for example, `LDAP.properties` or `AuthenticationSample.properties`, or `<your_custom_auth_module>.properties`.

When deploying your customized pluggable authentication module, you must make sure that the `.properties` file is copied or located in `<DSAME_root>/SUNWam/web-apps/services/WEB-INF/config/auth /default` directory, or the `auth/locale` directory. The authentication service will look for its configuration files in the `<DSAME_root>/SUNWam/web-apps/services/WEB-INF/config/auth /default` directory, by default. If the files are organization-specific, then it will in the `<DSAME_root>/SUNWam/web-apps/services/WEB-INF/config/auth /orgname` directory. If the files are organization and locale-specific, the authentication service will look for its configuration files in the `<DSAME_root>/SUNWam/web-apps/services/WEB-INF/config/auth /orgname/locale` directory.

Table 2-2 discusses the directives that can be included in a `.properties` file.

Table 2-2 The `.properties` File Directives

Directive	Description
SCREEN	Each SCREEN entry corresponds to one authentication state (authentication HTML page). The authentication module can set which screen is next, or it can allow the iPlanet Portal Server's `auth servlet` 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 is similar to a title for the page, and the text <xxx> appears at the top of the screen area provided for the auth module. Only one TEXT directive per SCREEN should be specified. If more than one is provided, then the last one is displayed.
TOKEN yyy	The TOKEN directive equates to the following HTML: `<P><STRONG>yyy</STRONG> <INPUT TYPE=TEXT NAME=TOKEN0>` The input field's name starts at TOKEN0 and increments with each TOKEN or PASSWORD directive specified per SCREEN.
PASSWORD zzz	The PASSWORD directive equates to the following HTML: <P><STRONG>yyy</STRONG> <INPUT TYPE="PASSWORD" NAME=TOKEN0> The input field's name starts at TOKEN0 and increments with each PASSWORD or TOKEN directive specified per SCREEN.
IMAGE image-path	The optional IMAGE directive allows authentication module writers to display a custom background image on each page.
HTML	The HTML parameter allows the module writer to use a custom HTML page for the authentication screens.
<REPLACE>	The REPLACE tag allows a module writer to substitute dynamic text for the accompanying text descriptions. This allows a module writer to dynamically generate challenges or passwords. Used in conjunction with the `setReplaceText()` method.

The specific directives included will depend on the requirements of the authentication method and the extent of customizing the appearance of the prompts and displays through the authentication process.

Each `SCREEN` entry corresponds to one authentication state or authentication HTML page. When an authentication session is invoked, one HTML page is sent for each state. The first state sends an HTML page asking the users to enter a token and a password. (Your authentication module can control the order; by default, it is sequential.) When the users submit the token and the password, the `validate()` method is called. The module gets the tokens, validates them, and returns them. The second page is then sent and the `validate()` method is again called.

If the module throws a `LoginException`, an authentication failed page is sent to the user. If no exception is thrown, which implies successful completion, the users are redirected to their default page. The `TIMEOUT` directive is used to ensure that the users respond in a timely manner. If the time between sending the page and the response is greater than the `TIMEOUT` value, a time-out page is sent.

When multiple pages are sent to the user, the tokens from a previous page may be retrieved by using the `getTokenForState` methods. Each page is referred to as a state. The underlying authentication module keeps the tokens from the previous states until the authentication is completed.

Each authentication session creates a new instance of the authentication Java class. The reference to the class is released once the authentication session has either succeeded or failed.

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

Code Example 2-2 Sample.properties File

```
SCREEN
TEXT This is a sample login page
TOKEN First Name
TOKEN Last Name

SCREEN
TIMEOUT 30
TEXT   You made it to page 2
PASSWORD Just enter any password

SCREEN
TIMEOUT 60

TEXT You made it past the first page
TOKEN  Enter <REPLACE>'s favorite beer
PASSWORD Enter <REPLACE>'s favorite wine

SCREEN
TEXT  4th page
PASSWORD any password
TOKEN anything here
```

The .properties file (for example, for the LDAP authentication module, see `LDAP.properties` in Code Example 2-7 on page 54) interacts with the authentication program, and displays a screen or multiple screens to a user in the process of authenticating the user. This might be a multi-step process for the user to authenticate, which may prompt the user for multiple inputs. Each screen corresponds to a step in the user authentication process. For example, when a user logs into the LDAP authentication module, it is the .properties file that provides the fields and text that appear on the authentication screens. For example, the `LDAP.properties` file describes text fields that describes the screen (such as "LDAP Authentication"), tokens (fields) that prompt the user to enter his/her userid, password prompt text, and timeout settings (for example, "120" might give the user 120 seconds to authenticate, after which it times out).

When writing a pluggable authentication module, you must follow the format specified for the .properties file. For example, the "TEXT" field specifies the title of the authentication HTML page. The "TOKEN" field (or keyword) specifies what displays on the authentication HTML screen or page. The "PASSWORD" field or keyword specifies what is displayed on the authentication HTML page. The "HTML" field or keyword can be used to specify that an entire HTML file be read in and displayed unmodified, instead of dynamically generating the HTML. (See Section "Understanding the screen.properties File" on page 36.)

Product Directories where .properties and Sample .java Files are Located

To see a sample .properties file and some sample authentication programs (`.java` files), go to the `<server_root>/samples/authentication/providers` directory.) The .properties files for the supported authentication modules (LDAP, Certificate-based, RADIUS, Membership, and Anonymous) are located in the `<DSAME_root>/SUNWam/web-apps/services/WEB_INF/classes` directory.

NOTE A TOKEN in a .properties file is like a keyword or field; it has no relation to an SSO token.

Every authentication module (for example, LDAP, RADIUS, Certificate-based, Membership (Self-registration), and Anonymous) has a .properties file. After a customization engineer creates a new application or service, he or she must copy the .properties file to the `<DSAME_root/web-apps/services/WEB-INF/config/auth/default` directory. (See Section "Writing and Integrating a Pluggable Authentication Module" on page 42 for information on how to integrate a custom authentication service.)

Use an existing service XML file to create your Custom Authentication XML

When writing a custom authentication service, you could typically use one of the existing internal authentication service XML files (for example, `amAuthLDAP.xml`) which is provided in DSAME, and create a copy which you can then customize with your custom authentication service's attributes. After loading in this new service XML file using the amadmin CLI tool, make sure you rename the XML file with the name of your custom authentication module.

amAuth.xml is Used for General Authentication Configuration

The `amAuth.xml` file defines the Core authentication service, which is the overall configuration file for authentication. It defines attributes that are used by the Core authentication service (which defines the overall authentication configuration).

When creating a custom authentication service, customization engineers must modify the `amAuth.xml` file to include their custom authentication module `.class` file (`com.iplanet.authentication.spi.AuthenticationSample.class`, for example) in the Authentication menu choices and in the Authenticator's list. (See Section "Writing and Integrating a Pluggable Authentication Module" on page 42.)

For some information on when a service developer would need to write a custom authentication service XML, see the Section "Do You need to create a Service XML for your custom Authentication Service" on page 34.

NOTE The Core authentication service (defined in `amAuth.xml`) defines overall authentication configuration. The administrator must always configure the Core authentication module in DSAME Console.

For information on configuring the various authentication modules supported, see the *iPlanet Directory Server Access Management Edition Administration Guide.*

writing and Integrating a Pluggable Authentication Module

The following sections discuss the requirements and recommendations to follow when writing a new authentication module.

Requirements and Recommendations

A pluggable authentication module for iPlanet DSAME must override certain methods in the `AuthenticationModule` class and the `AuthenticationModuleFactory` interface and should adhere to specific naming conventions and standards for easy integration.

NOTE For a list and description of the methods used to write the authentication module, see the JavaDocs at
`http://`*yourserver:port*`/docs/en_US/javadocs`

- Write your authentication module class which extends and subclasses methods in `com.iplanet.authentication.spi.AuthenticationModule`
- Override the `validate()`, `init()`, and `getUserTokenId()` methods

  The validate method replaces the input gathering method. Each time the user submits an HTML page, the `validate()` method will be called. In the method, authentication-specific routines are called. At any point in this method, if the authentication has failed, the module must throw a `LoginException`. If desired, the reason for failure can be an argument to the exception. This reason will be logged in the iPlanet Portal Server authentication log.

  `init()` should be used if the class has any specific initialization such as loading a JNI library.
  `init()` is called once for each instance of the class. Every authentication session creates a new instance of the class. Once a login session is completed the reference to the class is released.

  `getUserTokenId()` is called once at the end of a successful authentication session by the iPlanet Portal Server authentication server. This is the string the authenticated user will be known as in the iPlanet Portal Server environment. A login session is deemed successful when all pages in the `.properties` file have been sent and the module has not thrown an exception.
- Write your authentication module factory class which implements the `com.iplanet.authentication.spi.AuthenticationModuleFactory` interface

  You must also implement the AuthenticationModuleFactory interface when creating a pluggable authentication module. The AuthenticationModuleFactory interface defines a factory API which enables the authentication framework to obtain an instance of the corresponding authentication module. It creates a new instance of an AuthenticationModule object for an authentication module.This could be implemented by calling the corresponding constructor of your pluggable authentication module.

NOTE
- The newAuthenticationModule method is not synchronized.

The implementation of your pluggable authentication module should be thread-safe.

Recommendations

Naming the authentication module and factory classes you write in the following way allows (and forces) the module's class file to be installed with the other authentication modules provided with the iPlanet DSAME software:

If your authentication module class is named `<your_custom_auth_module>.java`, then name your authentication module factory class `<your_custom_auth_module>AuthenticationModuleFactory.java`

Compiling the Authentication Sample

Do the following to compile the Authentication Sample program (or a pluggable authentication module that you write at your site). The Authentication Sample is provided in the `<dsame_root>/samples/authentication/providers` directory:

Set Environment Variables

1. Set the following environment variables.

   These variables are used to run the `gmake` command. You can also set these variables in the `makefile`. The `Makefile` is located in the same directory (`<dsame_root>/SUNWam/samples/authentication/providers`) as the Authentication Sample program files.

   a. JAVA_HOME —Set this variable to the directory pathname where your JDK is installed. The JDK should be newer than JDK 1.2.2.

   b. CLASSPATH—Set this variable to the location of the `dai.jar` file, which is located in `<dsame_root>/SUNWam/web-apps/services/WEB-INF/lib`

   c. BASE_CLASS_DIR—Set this variable to the directory where all the sample compiled classes are located.

   d. JAR_DIR—Set this variable to the directory where the JAR files (.jar) of the Sample compiled classes will be created.

Run the Make Command

1. Go to the `<dsame_root>/SUNWam/samples/authentication/providers` directory and run

   ```
   make
   ```

Integrating the Authentication Sample program

Do the following to integrate, or deploy, the AuthenticationSample program (or a pluggable authentication module that you write):

1. Copy `<your_custom_auth_module>.jar` from the <JAR-DIR> directory to:

   `<dsame_root>/SUNWam/web-apps/services/WEB-INF/lib`

2. Copy `<your_custom_auth_module>.properties` file (or the `AuthenticationSample.properties` file if you using the sample files) to:

   `<dsame-root>/SUNWam/web-apps/services/WEB-INF/config/auth/default`

NOTE The `.properties` filename should be similarly named as `<your_custom_auth_module>.properties`; for example, `AuthenticationSample.properties`.

3. Modify `<dsame-root>/SUNWam/web-apps/services/WEB-INF/config/xml/amAuth.xml` to include your custom sample (or the `AuthenticationSample`) in the Authentication menu choices and in the Authenticators list in DSAME Admin Console as follows:

Code Example 2-3 Excerpt from amAuth.xml File

```
<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>
```

Code Example 2-3 Excerpt from amAuth.xml File

```
                  <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. In addition to modifying the `amAuth.xml` file, you might need to create a new service XML file for your custom authentication module.
5. Back up the iPlanetAMAuthService by using `db2ldif`:
   a. `cd <directory_install_root>/slapd-<hostname>`
   b. `./db2ldif -n userRoot -s "ou=iPlanetAMAuthService,ou=services,<root_suffix>"`
6. Delete the iPlanetAMAuthService entry using `amadmin`:

a. `/etc/init.d/amserver stop`

b. `cd <install-root>/SUNWam/web-apps/services/WEB-INF/bin`

c. `./amadmin -runAsDN uid=amAdmin,ou=People,<default_org>,<root_suffix> --password <password> --deleteService iPlanetAMAuthService`

7. Import the (modified) `amAuth.xml` file using the `amadmin` CLI tool:

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

8. Import the new service XML file for the pluggable authentication module that you have written, using amadmin:

   a. `./amadmin -runAsDN uid=amAdmin,ou=People,<default_org>,<root_suffix> --password <password> --schema <your_custom_auth_module>.xml`

NOTE The new service XML file should include only the attributes to be updated.

Also, the authentication module does not have to have a service XML file unless it has configuration attributes that will need to be manageable through the DSAME Admin Console. An example of configuration attributes that would need to be manageable through Admin Console are port numbers, defaults, and server locations.

9. Add `<your_custom_auth_module>.jar` file pathname (or the `AuthenticationSample.jar` file pathname) to the Web server JVM classpath:

   cd <install-root>/SUNWam/servers/https-<host>.<domain>/config

10. Modify the `jvm12.conf` file to add `<install-root>/SUNWam/web-apps/services/WEB-INF/lib/AuthenticationSample.jar` path to the JVM classpath.

Running the Authentication Sample program

Do the following steps to run the AuthenticationSample program:

1. Restart the DSAME server:

   ```
   <install-root>/SUNWam/web-apps/services/WEB-INF/bin/
   amserver start
   ```

2. Log in to the DSAME console by entering the URL

   ```
   http://<host>.<domain>:<port>/amserver/console
   ```

3. Select the User Management view.
4. Select your organization and select Services from the Show menu.
5. Click on DSAME Core Authentication properties.
6. Add AuthenticationSample from the Authentication menu.

   When you log in, you can choose either LDAP or your custom authentication module.

NOTE Do not de-select LDAP.

7. Click Submit to save changes and log out
8. Enter the URL

   ```
   http://<host>.<domain>:<port>/amserver/login
   ```

   and select your custom authentication module (or the AuthenticationSample) from the Authentication menu.

NOTE If you select your new authentication module and it keeps returning to the Authentication menu, check that your CLASSPATH variable setting is correct in the `jvm12.conf` file. Also check the `package.classname` in `amAuth.xml` file; make sure they match.

Also check the amAuth debug file for errors.

or

Enter the URL

```
http://<host>.<domain>:<port>/login?module=
your_custom_auth_module
```

Sample Code

The following samples show the form and content of the files associated with an authentication module:

- Sample Properties File
- Sample Authentication Module Source
- Sample Authentication XML File

Sample Properties File

There is a `Sample.properties` provided, which should be copied from:

`<DSAME_root>/SUNWam/samples/authentication/providers`

to

`<DSAME_root>/SUNWam/web-apps/services/WEB-INF/config/auth/default`

where `/opt` is the directory in which it is installed by default.

Code Example 2-4 AuthenticationSample.properties File

```
SCREEN
TEXT This is a sample login page
TOKEN First Name
TOKEN Last Name

SCREEN
TIMEOUT 30
TEXT   You made it to page 2
PASSWORD Just enter any password

SCREEN
TIMEOUT 60
TEXT You made it past the first page
TOKEN  Enter <REPLACE>’s favorite beer
PASSWORD Enter <REPLACE>’s favorite wine

SCREEN
TEXT  4th page
PASSWORD who cares
TOKEN anything here
```

Sample Authentication Module Source

The following samples are located in files named `AuthenticationSample.java` and `AuthenticationModuleFactory.java` which are located in:

```
<DSAME_root>/SUNWam/samples/authentication/providers
```

Code Example 2-5 Sample Java Module—AuthenticationSample.java

```
/*
* "$Id: AuthenticationSample.java,v 1.1 2001/08/21 02:16:21 uchil
Exp $"
* Copyright $Date: 2001/08/21 02:16:21 $ Sun Microsystems, Inc.
All Rights Reserved.
*
* Some preexisting portions Copyright C 2000 Netscape
Communications
* Corporation. All rights reserved.
*
* Sun, Sun Microsystems, and the Sun logo are trademarks or
registered
* trademarks of Sun Microsystems, Inc. in the United States and
other
* countries. Netscape and the Netscape N logo are registered
trademarks of
* Netscape Communications Corporation in the U.S. and other
countries. Other
* Netscape logos, product names, and service names are also
trademarks of
* Netscape Communications Corporation, which may be registered in
other
* countries.
*
* Federal Acquisitions: Commercial Software--Government Users
Subject to
* Standard License Terms and Conditions
*
* The product described in this document is distributed under
licenses
* restricting its use, copying, distribution, and decompilation.
No part of
* the product or this document may be reproduced in any form by
any means
* without prior written authorization of the Sun-Netscape
Alliance and its
* licensors, if any.
*
* THIS DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR
IMPLIED
```

Code Example 2-5 Sample Java Module—AuthenticationSample.java *(Continued)*

```
* CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
IMPLIED WARRANTY
* OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
NON-INFRINGEMENT,
* ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE
HELD TO BE
* LEGALLY INVALID.
*/

package com.iplanet.am.samples.authentication.providers;

import java.util.*;
import com.iplanet.authentication.spi.*;

public class AuthenticationSample extends AuthenticationModule {

    private String userTokenId;
    private String firstName;
    private String lastName;

    public AuthenticationSample() throws AuthenticationException{
    System.out.println("AuthenticationSample()");
    }

    public void init() throws AuthenticationException {
    System.out.println("AuthenticationSample initialization");
    }

    public void validate() throws AuthenticationException {

    int currentState = getCurrentState();

    if (currentState == 1) {
        firstName = getToken(1);
        lastName = getToken(2);
        if (firstName.equals("") || lastName.equals("")) {
    throw new AuthenticationException("names must not be empty");
        }
        return;
    }
    else if (currentState == 2) {
        String pass = getToken(1);
        System.out.println("Replace TExt first: " + firstName + "
last: " + lastName);
        setReplaceText(1, firstName);
        setReplaceText(2, lastName);
        return;
    }
    else if (currentState == 3) {
        String[] tokens = getAllTokens();
        for (int i=0; i<getNumberOfTokens(); i++) {
    System.out.println("Token-> " + tokens[i]);
        }
        return;
```

Code Example 2-5 Sample Java Module—AuthenticationSample.java *(Continued)*

```
    }
    else if (currentState == 4) {
        String[] tokens = getAllTokensForState(1);
        for (int i=0; i<getNumberOfTokensForState(1); i++) {
    System.out.println("Token-> " + tokens[i]);
        }
    }

    userTokenId = firstName;
    }

    public String getUserTokenId() {
    return userTokenId;
    }

    private static final String sccsID = "$Id:
AuthenticationSample.java,v 1.1 2001/08/21 02:16:21 uchil Exp $
Sun Microsystems, Inc.";
}
```

Code Example 2-6 AuthenticationSampleAuthenticationModuleFactory.java

```
/*
* "$Id: AuthenticationSampleAuthenticationModuleFactory.java,v
1.1 2001/08/21 02:17:46 uchil Exp $"
* Copyright $Date: 2001/08/21 02:17:46 $ Sun Microsystems, Inc.
All Rights Reserved.
*
* Some preexisting portions Copyright C 2000 Netscape
Communications
* Corporation. All rights reserved.
*
* Sun, Sun Microsystems, and the Sun logo are trademarks or
registered
* trademarks of Sun Microsystems, Inc. in the United States and
other
* countries. Netscape and the Netscape N logo are registered
trademarks of
* Netscape Communications Corporation in the U.S. and other
countries. Other
* Netscape logos, product names, and service names are also
trademarks of
* Netscape Communications Corporation, which may be registered in
other
* countries.
*
```

Code Example 2-6 AuthenticationSampleAuthenticationModuleFactory.java

```
* Federal Acquisitions: Commercial Software--Government Users
Subject to
* Standard License Terms and Conditions
*
* The product described in this document is distributed under
licenses
* restricting its use, copying, distribution, and decompilation.
No part of
* the product or this document may be reproduced in any form by
any means
* without prior written authorization of the Sun-Netscape
Alliance and its
* licensors, if any.
*
* THIS DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR
IMPLIED
* CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY
IMPLIED WARRANTY
* OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR
NON-INFRINGEMENT,
* ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE
HELD TO BE
* LEGALLY INVALID.
*/

package com.iplanet.am.samples.authentication.providers;

import com.iplanet.authentication.spi.*;

/*
 * Defines the Authentication Sample module factory
 */
public class AuthenticationSampleAuthenticationModuleFactory
    implements AuthenticationModuleFactory
{
    /*
     * Returns an Authentication Sample module instance
     */
    public AuthenticationModule newAuthenticationModule()
    throws AuthenticationException
    {
       return (AuthenticationModule) new AuthenticationSample();
    }
}
```

Sample XML Files

When creating a new service XML file, you can copy an existing authentication service XML file, and alter it as needed for your new customized authentication service. For information on writing a new service, see Chapter 5, “Understanding DSAME XMLs and DTDs".

When creating a new service, you must modify the amAuth.xml file (Core authentication) as described in Section “amAuth.xml is Used for General Authentication Configuration” on page 42.

Resource Bundle .properties File

Every service in DSAME has a corresponding resource bundle .properties file (for an example, see Code Example 2-7 on page 54).

NOTE The resource bundle .properties file is different from the screen .properties file. Typically, each service in DSAME, whether internal or external, must have a resource bundle .properties file, and a screen .properties file.

Each service in DSAME (for example, the LDAP authentication service—described in `dpAuthLDAP.xml`) knows the name of its properties file. The properties in each service’s resource bundle properties file are retrieved using the ResourceBundle class. Note that every service’s resource bundle .properties file contains keys (such as “a1”, “a2”, etc.) which map to the actual fields as they will display on each service’s page in Admin Console. These index, or localization, keys are described in each service’s XML file (for example, `amAuthLDAP.xml`, etc.)

For example, for LDAP Authentication, all of these localization keys, or index keys, are defined in the `amAuthLDAP.xml` file, and similarly for each internal DSAME service (logging, session, authentication, etc.) and each external new service that a customization engineer might add to DSAME.

Code Example 2-7 amAuthLDAP.properties File

```
LDAPex=Unknown LDAP Exception
UPerror=Both UserId and Password Required
classpathError=Cannot find the Class, check classpath
InvalidUP=Invalid userId and password, please try again.
NoUser=Cannot find userId
NoServer=cannot contact server
Naming=naming error
```

Code Example 2-7 amAuthLDAP.properties File

```
iplanet-am-auth-ldap-service-description=LDAP
PasswordExp=Password Expires In
PasswdMismatch=New Passwords don’t match Reenter
UPsame=username and password are same
PInvalid=Cureent Password Entered Is Invalid
PasswdSame=Password should not be same
PasswdMinChars=Password should be atleast 8 characters
a1=LDAP Server and Port
a2=DN to Start User Search
a3=DN for Root User bind
a4=Password for Root User Bind
a5=Search Filter
a6=User Entry Naming Attribute
a7=Search Scope
a8=Enable SSL to LDAP Server
a9=Authentication Level
```

Chapter 3

HTML Templates

This chapter provides information on how to customize the authentication login, logout, and timeout pages for different organizations in DSAME console, and gives some information on how authentication templates work.

Setting up Login Pages for Different Organizations

You can edit HTML templates to make substantive changes to the layout or design of pages, or to add extra functionality, beyond the services possible through the DSAME console.

NOTE Strong HTML skills as well as a thorough understanding of Web servers are required to edit the template files.

It is recommended that you make backups of the templates files before making modifications to them; then you can restore your files from the backups.

Alternatively, if a template file is corrupted, you can restore the original files from the iPlanet Directory Server Access Management Edition CD-ROM to recover and gain access to the system.

How Authentication Templates Work

HTML template files control the layout of the iPlanet Directory Server Management Access Edition console pages and of the other screens that users see. The templates are located on the iPlanet Directory Server Management Access Edition in the directory:

dsame_root/SUNWam/web-apps/services/WEB-INF/config/auth/default

Templates for Customizing the Authentication Pages

These templates allow you to customize the login, logout, and time-out screens.

In the *DSAME_root*/SUNWam/web-apps/services/WEB-INF/config/auth/default directory, there are .html files that control the overall appearance and .properties files that control the sequence of prompts and the exchange of information between the user and the authentication module.

The login pages come from a set of template HTML files. The default set of these is located at *DSAME_root*/SUNWam/web-apps/services/WEB-INF/config/auth/defaults directory.

To Customize a Login for Different Organizations:

1. Go to the server machine (do the same to all server machines if there are multiple servers).

   cd *DSAME_root*/SUNWam/web-apps/services/WEB-INF/config/auth

2. Create a directory with the name of the organization; this should be a DN (distinguished name), and should be the same name that appears in the DSAME console.

3. Copy all the .properties and .html files (and .gif files, if they exist) into that directory.

4. Customize the files in that directory for that organization.

Any organization that does not have its own directory of templates will use the default set in:

DSAME_root/SUNWam/web-apps/services/WEB-INF/config/auth/default

For example, if there are three organizations—org1, org2, org3—and you are customizing the login for org1, the directories will look like this:

DSAME_root/SUNWam/web-apps/services/WEB-INF/config/auth/default

DSAME_root/SUNWam/web-apps/services/WEB-INF/config/auth/org1

Both would contain a full set of the properties, html and gif files. The login to org1 would use the set in *DSAME_root*/SUNWam/web-apps/services/WEB-INF/config/auth/org1/, and the other organizations would use the default set in *DSAME_root*/SUNWam/web-apps/services/WEB-INF/config/auth/default

The HTML template files are described in Table 3-1.

Table 3-1 HTML Template Files

File Name	Description
login_menu.html	Is sent when more than one authentication module is configured. This gives the iPlanet Directory Server Access Management Edition end user a choice of which module to use for authentication. The text <subst data="rows">No menu?</subst> *must* be somewhere in the document. It generates a list of URLS to the authentication modules.
login_fail_template.html	Is sent when authentication has failed. This page contains no required sections.
login_reauth_menu.html	Is sent when a DSAME end user's session has been inactive for the time set in the DSAME console. It contains a link for re-authentication. Do not change the Javascript in this page.
login_template.html	Is sent for individual authentication modules such as RADIUS. The seven subset text segments must remain after modification. This page is also sent when logging in to the iPlanet Directory Server Access Management Edition Administration Console.
logout.html	Is called after the DSAME end user selects the logout link on the DSAME DSAME console. it contains no required sections.
login_timeout_template.html	Is called during an authentication session if the iPlanet Directory Server Access Management Edition end user does not submit the login form within the specified time. It has no required sections.
invalidPassword.html	Error page for invalid password length in self-registration page.
login_denied.html	Error page if user does not have a profile entry in this DSAME installation.
login_fail_admin.html	Login to DSAME console failed.
login_menu.html	Template used to display user login page when multiple authentication modules are enabled for an organization.

Table 3-1 HTML Template Files

File Name	Description
`login_menu_modules.html`	Used for single authentication module within login_menu.html
`login_prompt.html`	When user-based authentication is enabled, this page is used to ask for user login id
`login_reauth_admin.html`	Administration Console session has expired.
`login_template.html`	Template for the login page
`login_timeout_admin.html`	Timeout page for login to DSAME console
`login_timeout_template.html`	Login timeout template

Table 3-1 HTML Template Files

File Name	Description
`logout_admin.html`	User logout from DSAME console
`membership.html`	Self-registration login page
`missingReqField.html`	Error page when user is not allowed to use this authentication module.
`module_denied.html`	Error page when user is not allowed to use this authentication module
`noConfirmation.html`	Error page if no confirmation password entered in self-registration page
`noPassword.html`	Error page if no password entered in self-registration page
`noUserName.html`	Error page if no user name entered in self-registration pgae
`noUserProfile.html`	Error page if no matching user found when using self-registration to log in.
`org_inactive.html`	Error page if the matching user found when using self-registration to log in
`password_mismatch.html`	Error page if confirmation password does not match in self-registration page
`register.html`	Self-registration page
`session_timeout.html`	Error page if user session times out
`userExists.html`	Error page when trying to register a user that already exists in self-registration module
`userPasswordSame.html`	Error page when user enters a password that is the same as user ID in self-registration page
`user_inactive.html`	Error page if user login is disabled (not allowed to login)
`wrongPassword.html`	Error page if user enters invalid password when using self-registration module to log in

Chapter 4

Single Sign-On

This chapter describes how to use the public Single Sign-On APIs to create a custom single sign-on solution. It contains the following sections:

- How SSO Uses Cookies
- How SSO Uses Tokens
- Overview of Web-Based Single Sign On (SSO) APIs
- Using the SSO Samples
- SSOTokenSampleServlet.java File
- SampleTokenListener.java
- SSOTokenSampleServlet.java File

Introduction to the Single Sign-On Solution

Every business has resources and services that the business wants to protect. When users want to access these resources, they must authenticate themselves to get access to those resources. The user could be a human being or a service itself accessing a different service. If the users authenticate successfully, and if they are authorized to access those resources, the users are given access to those resources. When a user accesses several resources protected by authentication and authorization policies, it is necessary that a user authenticate every time a protected resource is accessed. The Single Sign-On feature is a solution that enables a user from having to repeatedly authenticate himself or herself to access

multiple applications and services. Providing an SSO solution means that the user will have to authenticate once only. This means that successive attempts by a user to access protected resources will not require the user to provide authentication information to get access to each resource.

Web users are typically required to use a separate password to authenticate themselves to each server they need to access during the course of their work. Multiple passwords are an ongoing headache for both users and system administrators. Users have difficulty keeping track of different passwords, tend to choose poor ones, and tend to write them down in obvious places. Administrators must keep track of a separate password database on each server and deal with potential security problems related to the fact that passwords are sent over the network routinely and frequently.

Solving this problem requires some way for a user to log in only once using a single password, and then get authenticated access to all servers that user is authorized to use—without sending any passwords over the network. This capability is known as Single Sign-On (SSO).

The Web-based Single Sign-On feature in DSAME 5.0 provides or enables the following:

- The Single Sign-On component or module provides Java interfaces for applications to participate in the Single Sign-On (SSO) solution.
- The SSO interfaces also include provisions for applications to register callback listeners, which can be invoked when an SSO token is destroyed
- An adapter servlet may be necessary to integrate non-web applications into DSAME Single Sign-On.
- Single Sign-On and Authentication support authentication levels.

How SSO Uses Cookies

An encrypted cookie is an information packet generated by the Web Server. Note that a cookie cannot cross domains. (This is a security requirement.) For example, a cookie generated for amazon.com cannot be used for another domain such as fatbrain.com.

A cookie maintains information regarding a user. However, having a cookie does not necessarily imply that the user is authenticated.

The Single Sign-On service and the Authentication service can be thought of as part of the Web Server. In the DSAME 5.0 implementation, the cookie is generated by the Single Sign-On service and set by the Authentication service which, again, is part of the Web server.

How SSO Uses Tokens

The SSO service generates an SSO token using a secure random number generator, then sends it to the Authentication component. The Authentication component then requests a token from the Session component. In DSAME 5.0, an SSO token is inserted into a cookie and sent back to the web browser by the Authentication service. You could consider that a valid SSO token represents an authenticated user.

The SSO APIs in DSAME 5.0 can be used to create SSO tokens after authentication to DSAME has been performed. Multiple tokens created using the SSO API for a single user will point to the same data internally.

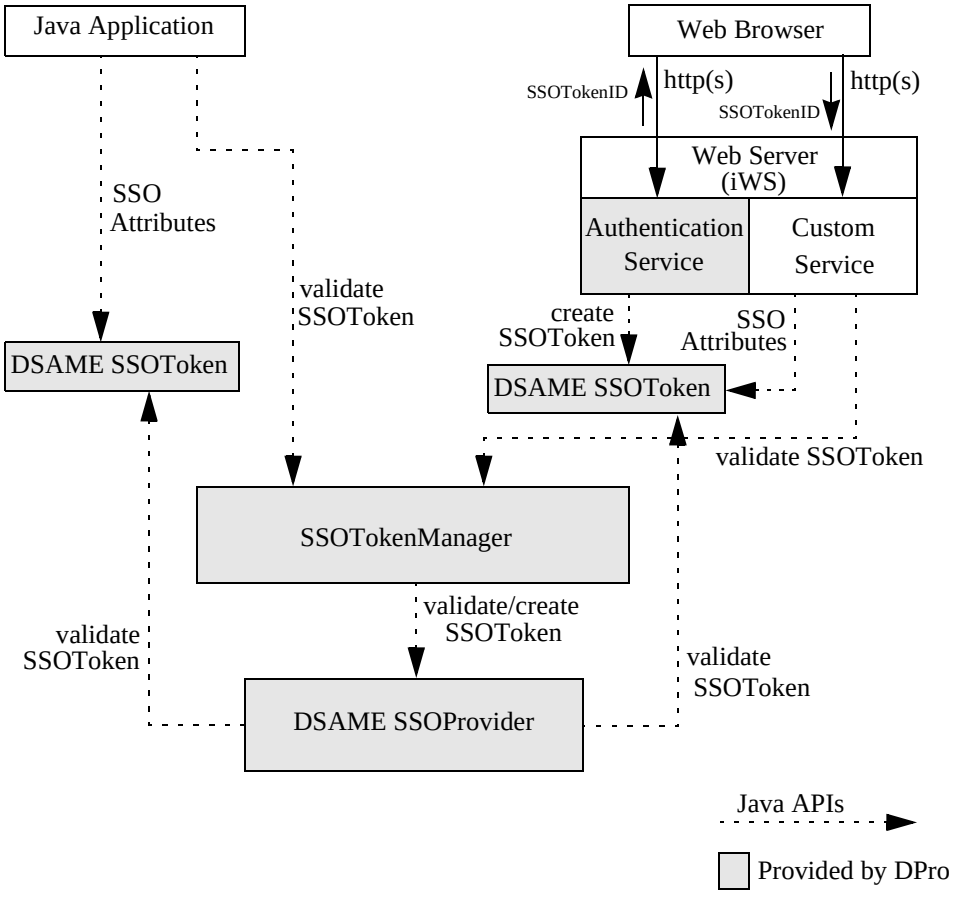
Overview of Web-Based Single Sign On (SSO) APIs

DSAME provides a Single Sign-On solution primarily for web-based applications, although it can be extended to any Java-based application, as well as non-Java applications. The SSO solution provides a mechanism by which users need to authenticate only once, and then can access multiple web-based applications without having to re-authenticate. Additionally, it provides interfaces for applications to store generic key-value pairs and to register callback listeners which will be invoked when the SSO token is destroyed.

After a user has been authenticated, it is possible to get an SSO token (either through Java interfaces, or from HTTP headers). This SSO token is the basis for providing a Single Sign-On solution. All DSAME's services and interfaces (except for authentication) need a valid SSO token to process the request. Other applications wishing to participate in the SSO solution must use the SSO token to validate the user's identity.

DSAME's SSO component mainly provides Java interfaces (that is, Java SDK) for applications to participate in the SSO solution. The SSO provides a federated architecture by which different single-sign-on solutions can be plugged in. Figure Figure 4-1 on page 66 shows the Java SDK architecture of SSO.

Figure 4-1 SSO SDK Architecture



Overview of SSO Classes/Interfaces

This section provides some overview information of the SSO classes and interfaces. Some of the SSO classes discussed are shown in Figure 4-1 on page 66.

The main class is *SSOTokenManager*, which is also the only concrete class in the Single Sign-On component. It provides methods to create, get, and validate SSO tokens. It is a *final* class and is a singleton. Other classes *SSOToken*, *SSOTokenID*, and *SSOProvider* are Java interfaces. Additionally, the SSO SDK provides *SSOTokenListener* and *SSOTokenEvent* to support notification when SSO tokens are invalidated.

The *SSOTokenManager* maintains a configuration database of the valid SSO providers (that is, valid implementations for *SSOProvider*, *SSOToken* and *SSOTokenID*).

The *SSOTokenManager* constructor will try to find the provider JAR files, load them, then find the provider mainclass, instantiate it, and store it in the provider (SSOProvider). Providers can be configured using the `providerimplclass` Java property (which is stored in the `<dsame_root>/web-apps/services/WEB-INF/classes/SSOConfig.properties` file). This property must be set to the complete (absolute) package name of the main class of the provider. The main class MUST implement the `com.iplanet.sso.SSOProvider` interface and MUST have a public no-arg default constructor.

A request to *SSOTokenManager* gets delegated to one of the providers (*SSOProvider*). Thus, the *SSOProvider* class performs the bulk of the function of *SSOTokenManager*. The *SSOTokenID* is a string representation of *SSOToken*.

The *SSOToken* class represents a “single sign-on” (SSO) token. It contains SSO token-related information such as authentication method used for authentication, authentication level of the authentication method, hostname of the client that sent the request (browser). It also contains session-related information such as maximum session time, maximum session idle time and session idle time.

NOTE. In DSAME 5.0, you cannot integrate your own SSO provider. This may be supported in a future release.

The *SSOTokenID* class is used to identify an SSOToken object. Additionally, the SSOToken ID string contains a random number, an SSO server host, and an SSO 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* (not the SSO token object) can be communicated from one servlet to another in one of the following ways:

- as a cookie in *http* header; or
- *SSOTokenListener* interface needs to be implemented by the applications to receive SSO token events.

The SSOTokenEvent class represents an SSO token event. A token is granted when a change in the state of the token occurs. The SSO token event represents a change in SSOToken. The following are possible SSO token event types: SSO_TOKEN_IDLE_TIMEOUT, SSO_TOKEN_MAX_TIMEOUT, and SSO_TOKEN_DESTROY.

The SSO token provides a listener mechanism for applications that need notification when the SSO token expires. The SSO token could expire because it could have reached its maximum session time, or idle time, or an administrator could have terminated the session. Applications wishing to be notified must register a callback object (which implements the *SSOTokenListener* interface) with the SSO token. The 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 for the SSO token to expire.

NOTE The DSAME 5.0 SSO APIs cannot be used in a multi-JVM environment, such as for iPlanet Application Server.

The *SSOException* class is thrown when there are errors related to SSOToken operations.

For detailed information on the public Single Sign-On APIs in this release, see the Javadocs, which are located in the `/opt/SUNWam/web-apps/services/docs/en_USjavadocs` directory. (The directory name may vary, depending on the language version that has been installed.)

SSO Feature Intended for SSO Client Applications

The primary purpose of the SSO APIs provided in DSAME 5.0 is for SSO client applications. SSO client applications can include any service or application that a service or application developer wants to make use of the Single Sign-On feature. These SSO client applications use the SSO token to validate Single Sign-On.

The SSO service generates the SSOToken for a user after the user is authenticated. Once the token is generated, it is carried along with the user as the user moves around the web. When that user tries to access any application or service (that is “SSO-aware” or “SSO-enabled”), such applications or services use the token to validate that user has already signed in.

You can use the the SSO service in DSAME 5.0 without any configuration or customization.

Public SSO Classes/Interfaces

The following is a list of the public SSO interfaces that you can use to create and customize your applications and services. Refer to the Javadocs for more information on functionality and how to use these interfaces to implement your own SSO client applications and services in DSAME 5.0. (The Javadocs are located in the `/opt/SUNWam/web-apps/services/docs/en_US/javadocs` directory. (The directory name may vary, depending on the language version that has been installed.)

- SSOTokenManager
- SSOToken
- SSOTokenListener
- SSOTokenEvent
- SSOTokenID
- SSOException

Implementing an SSO Solution

To implement an SSO solution (Single Sign-On service or application) there must be a way to keep track of user identification after the user is authenticated. The user identification will contain information such as:

- user's name
- authentication method by which the user is authenticated
- authentication level of the authentication method used

You can use this information to get the identity of the user and to give seamless access to the protected web resources.

DSAME provides a set of SSO Java API for the purpose of implementing your own SSO solution. Using these APIs, you can obtain the identity of the user and get authentication information related to the user. The applications can use this information to determine whether to provide access to a resource that a user or device requests access to. For overview information, see “Overview of SSO Classes / Interfaces,” on page 66. Also refer to the Javadocs for the SSO APIs. The Javadocs are located in the `/opt/SUNWam/web-apps/services/docs/en_USjavadocs` directory. (The directory name may vary, depending on the language version that has been installed.)

For example, say a user authenticates to a site `http://www.sun.com/SunStore` successfully. The user can later access another protected page `http://www.sun.com/UpdateMyInfo`. The UpdateMyInfo application will need to authenticate the user again to verify the identity of the user. Instead, this application can use the SSO APIs to determine if the user accessing the UpdateMyInfo application is already authenticated or not. If the SSO API methods indicate that the user is valid and has been authenticated already, then access to this page can be given directly without the user needing to authenticate again. Otherwise, the user may be prompted to authenticate again.

As an example, service developers can use the SSO Java APIs in the following way to determine if the user is already authenticated. (Additionally, the SSO APIs can be used to do things like perform a query on the token for information such as hostname, IP address, idle time, etc.) Refer to Code Example 4-1 on page 70.

Code Example 4-1 SSO Code Sample To Determine If User Is Already Authenticated

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

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

            /* use isValid to method to check if the token is valid or not
             * this method retuns true for valid token, false otherise
             */
            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());
            }
            /* let us try to validate the token again, with another method
             * if token is invalid, this method throws excpetion
             */
            manager.validateToken(token);

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

Code Example 4-1 SSO Code Sample To Determine If User Is Already Authenticated *(Continued)*

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

            /* print the string representation of the token */

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

            /* let us set some 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");
            /* let us 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, you might find it more efficient and convenient to use `SSOTokenManager.validateToken(token)` than `SSOTokenManager.isValidToken(token)`. `SSOTokenManager.validToken(token)` throws an SSOException when the token is invalid, thus terminating the method execution right away.

The SSO Java APIs can be used only for Web-based SSO. The APIs are not intended to be used in standalone existing non-Web applications to provide single-sign-on functionality. For example, you cannot use the SSO functions to modify an existing standalone Java application to participate in SSO.

It is also possible to get the SSO token if the SSO token ID string is passed to the applications. The example in that case would be something like this:

Code Example 4-2 Code Sample To Get SSO Token If SSO Token ID Is Passed To Applications

```
            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);

            /* let us set some 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");
            /* let us 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);
        }
    }
```

The applications may also listen for SSO Token events. It is possible that while a user is using an application, an SSO Token may become invalid. There may be several reasons why this may happen. The user's access time out because of the maximum time limit for a user's continuous access of resources. It could become invalid if the user fails to log out of an application and the idle time-out has expired. The application must be informed of such events so that it can terminate the user's access to the application when the tokens become invalid.

The SSO APIs can be used in the following way to get the SSO Token events.

Code Example 4-3 Code Sample To Register For SSOToken Events

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

where SampleTokenListener is a class defined as follows:

Code Example 4-4 Code Sample Showing SampleTokenListener Class Defined

```
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());
        }
    }
}
```

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. See “Overview of SSO Classes / Interfaces,” on page 66 for information on the SSO classes and interfaces’ behavior in this release.

Also see the SSO Javadocs for more details on the SSO APIs provided in the DSAME 5.0 release. The Javadocs are located in the /opt/SUNWam/web-apps/services/docs/en_USjavadocs directory. (The directory name may vary, depending on the language version that has been installed.)

Using the SSO Samples

A Readme file and some SSO samples are provided to assist service or application developers in writing, compiling, and running an SSO sample service. Also, there is a Readme.html which provides some information on how to compile and run the SSO sample application:

- <dsame_root>/samples/sso/Readme.html
- <dsame_root>/samples/sso/SSOTokenSample.java
- <dsame_root>/samples/sso/SSOTokenSampleServlet.java
- <dsame_root>/samples/sso/SampleTokenListener.java

See the information in this section for information on writing and integrating the SSO sample application.

Compiling and Running the SSO Sample Application

This section provides some information on how to use the SSO sample files provided to write, compile, and run an SSO sample application or service.

Additionally, there is a Readme.html file in the <dsame_root>/samples/sso directory that provides some instructions on how to compile and run the SSO sample applications provided. The following three SSO sample files are provided:

- SSOTokenSample.java
- SSOTokenSampleServlet.java

- `SampleTokenListener.java`

Follow the steps and information in these sections to set environment variables, compile the sample programs, and restart the iPlanet DS and Web servers.

Setting Environment Variables for SSO Sample Programs

1. Set the following environment variables.

   These environment variables are used to run the `make` command. You can also set these variables in the `Makefile`. This Makefile is located in the same directory as the sample files (`<dsame_root>/samples/sso`).

   a. CLASSPATH—Reference to all the JAR files located in the `<dsame_root>/SUNWam/web-apps/services/WEB-INF/lib` directory.

   b. JAVA_HOME —Set this variable to the directory where your JDK is installed.

NOTE It is not necessary that you use the iPlanet Web Server version of JDK/JRE. You can use any version of JDK, provided that it is newer than version 1.2.2.

---

   c. BASE_CLASS_DIR—Directory where all the Sample compiled classes will be located.

   d. JAR_DIR—Directory where the JAR files of the sample classes will be created.

Run the gmake Command to Compile the Sample Programs

1. Go to the directory

   ```
   <dsame_root>/SUNWam/samples/sso
   ```

   then run:

   ```
   gmake
   ```

2. Go to the directory that the JAR_DIR variable is set to. Copy the `SSOSample.jar` file to the `<dsame_root>/SUNWam/web-apps/services/WEB-INF/lib` directory.

Register the Sample Servlet

1. Register the Sample servlet. To do this, insert the lines below at the end of the file:

   `<dsame_root>/SUNWam/web-apps/services/WEB-INF/web.xml`

Code Example 4-5 Lines that register Sample servlet to be added to web.xml File

```
        <servlet>
                <description>SSOTokenSampleServlet</description>
                <servlet-name>SSOTokenSampleServlet</servlet-name>

<servlet-class>com.iplanet.am.samples.sso.SSOTokenSampleServlet</servlet-class
>
        </servlet>
        <servlet-mapping>
                <servlet-name>SSOTokenSampleServlet</servlet-name>
                <url-pattern>/SSOTokenSampleServlet</url-pattern>
        </servlet-mapping>
```

Restart the DSAME server (and iDS and Web servers)

1. Restart the server:

   `amserver start`

   This restarts the DSAME server, the Directory server, and the Web server.

2. Assign a policy to the SSOTokenSampleServlet; otherwise, you will get an "access denied" page. (You can do this through Admin Console.)

3. Log in to the DSAME console by typing in the browser:

   `<protocol>://<host>:<port>/SSOTokenSampleServlet`

   You may have to add a deployment prefix "amserver".

Your sample program should display the output in the browser. In the URL shown in Step 3 above `<host>` must be a fully-qualified name; for example, `salida10.red.iplanet.com`.

SSOTokenSampleServlet.java File

The SSOTokenSampleServlet.java file is for creating an SSOToken from an HTTP request object.

Following is the `SSOTokenSampleServlet.java` file. Refer to "Overview of SSO Classes / Interfaces," on page 66 for some information on the SSO classes and interfaces' behavior. Also refer to the Javadocs, which are located in the `/opt/SUNWam/web-apps/services/docs/en_USjavadocs` directory. (The directory name may vary, depending on the language version that has been installed.)

Code Example 4-6 SSOTokenSampleServlet.java File

```
/**
 * $Id: SSOTokenSampleServlet.java,v 1.2 2001/10/25 00:56:19 jfn Exp $
 * Copyright 2001 Sun Microsystems, Inc. Some preexisting
 * portions Copyright 2001 Netscape Communications Corp.
 * All rights reserved. Use of this product is subject to
 * license terms. Federal Acquisitions: Commercial Software --
 * Government Users Subject to Standard License Terms and
 * Conditions.
 *
 * Sun, Sun Microsystems, the Sun logo, and iPlanet are
 * trademarks or registered trademarks of Sun Microsystems, Inc.
 * in the United States and other countries. Netscape and the
 * Netscape N logo are registered trademarks of Netscape
 * Communications Corporation in the U.S. and other countries.
 * Other Netscape logos, product names, and service names are
 * also trademarks of Netscape Communications Corporation,
 * which may be registered in other countries.
 */

package com.iplanet.am.samples.sso;

import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.iplanet.sso.*;

/**
 * This Sample serves as a basis for using SSO API. It demonstrates creating
 * a SSO Token, calling various methods from the token, setting up event
 * listeners and getting called on event listeners. For detailed info refer to
the
 * Readme.txt
 *
 * @see com.iplanet.sso.SSOToken
 * @see com.iplanet.sso.SSOTokenID
 * @see com.iplanet.sso.SSOTokenManager
 * @see com.iplanet.sso.SSOTokenEvent
 * @see com.iplanet.sso.SSOTokenListener
 */

public class SSOTokenSampleServlet extends HttpServlet {

    public void init() throws ServletException {
    }
```

Code Example 4-6 SSOTokenSampleServlet.java File

```
    public void doGet(HttpServletRequest request, HttpServletResponse
response){

        try {
            ServletOutputStream out = response.getOutputStream();

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

            /* use isValid to method to check if the token is valid or not
             * this method retuns true for valid token, false otherise
             */
            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());
            }
            /* let us try to validate the token again, with another method
             * if token is invalid, this method throws excpetion
             */
            manager.validateToken(token);

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

            String id = tokenId.toString();

            /* print the string representation of the token */

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

            /* let us set some 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);
```

Code Example 4-6 SSOTokenSampleServlet.java File

```
            out.println("SSO Token Validation test Succeeded");
            /* let us 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();
        }
    }
}
```

SampleTokenListener.java

Following is the SampleTokenListener.java file. Refer to “Overview of SSO Classes / Interfaces,” on page 66 for some information on the SSO APIs’ behavior; also refer to the Javadocs for more information. (The Javadocs are located in the `/opt/SUNWam/web-apps/services/docs/en_USjavadocs` directory. (The directory name may vary, depending on the language version that has been installed.)

Code Example 4-7 SampleTokenListener.java File

```
/**
 * $Id: SampleTokenListener.java,v 1.2 2001/10/25 00:56:19 jfn Exp $
 * Copyright 2001 Sun Microsystems, Inc. Some preexisting
 * portions Copyright 2001 Netscape Communications Corp.
 * All rights reserved. Use of this product is subject to
 * license terms. Federal Acquisitions: Commercial Software --
 * Government Users Subject to Standard License Terms and
 * Conditions.
 *
 * Sun, Sun Microsystems, the Sun logo, and iPlanet are
 * trademarks or registered trademarks of Sun Microsystems, Inc.
 * in the United States and other countries. Netscape and the
 * Netscape N logo are registered trademarks of Netscape
 * Communications Corporation in the U.S. and other countries.
 * Other Netscape logos, product names, and service names are
 * also trademarks of Netscape Communications Corporation,
 * which may be registered in other countries.
 */

package com.iplanet.am.samples.sso;
```

Code Example 4-7 SampleTokenListener.java File *(Continued)*

```
import com.iplanet.sso.*;

/**
 * This is the SSO token listener class to listen for sso token events.
 * Token events are received when the token state changes. The ssoTokenChanged
 * is the method that gets called when the event arrives.
 */
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());
        }
    }
}
```

SSOTokenSampleServlet.java File

The SSOTokenSampleServlet.java file can be used to create an SSOToken from an SSOTokenID string.

NOTE A sample .java file is provided to create an SSOToken sample servlet in the DSAME 5.0 release. You should note, however, that the Single Sign-On feature in DSAME 5.0 is primarily web-based.

Following is the SSOTokenSample.java file. Refer to earlier in this chapter for some brief explanations on how part of this sample code works; also refer to the Javadocs for more information.

Code Example 4-8 SSOTokenSample.java File

```
/**
 * $Id: SSOTokenSample.java,v 1.2 2001/10/25 00:56:19 jfn Exp $
 * Copyright 2001 Sun Microsystems, Inc. Some preexisting
 * portions Copyright 2001 Netscape Communications Corp.
 * All rights reserved. Use of this product is subject to
 * license terms. Federal Acquisitions: Commercial Software --
 * Government Users Subject to Standard License Terms and
 * Conditions.
 *
 * Sun, Sun Microsystems, the Sun logo, and iPlanet are
 * trademarks or registered trademarks of Sun Microsystems, Inc.
 * in the United States and other countries. Netscape and the
 * Netscape N logo are registered trademarks of Netscape
 * Communications Corporation in the U.S. and other countries.
 * Other Netscape logos, product names, and service names are
 * also trademarks of Netscape Communications Corporation,
 * which may be registered in other countries.
 */

package com.iplanet.am.samples.sso;

import java.net.*;
import com.iplanet.sso.*;

/**
 * This sample serves as a basis for using SSO API. It demonstrates creating
 * a SSO Token, calling various methods from the token, setting up event
 * listeners and getting called on event listeners. Refer to the Readme.txt for
 * detailed info on how to use this sample.
 *
 * @see com.iplanet.sso.SSOToken
 * @see com.iplanet.sso.SSOTokenID
 * @see com.iplanet.sso.SSOTokenManager
 * @see com.iplanet.sso.SSOTokenEvent
 * @see com.iplanet.sso.SSOTokenListener
 */

public class SSOTokenSample {
    public static void main(String[] args) {

        try {
```

Code Example 4-8 SSOTokenSample.java File

```
            SSOTokenManager manager = SSOTokenManager.getInstance();

            SSOToken token = manager.createSSOToken(args[0]);

            /* use isValid method to check if the token is valid or not
             * this method retuns true for valid token, false otherise
             */
            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();
                System.out.println("SSOToken host name: " + host);
                System.out.println("SSOToken Principal name: " +
                                   principal.getName());
                System.out.println("Authentication type used: " + authType);
                System.out.println("IPAddress of the host: " +
                                   ipAddress.getHostAddress());
            }
            /* let us try to validate the token again, with another method
             * if token is invalid, this method throws an excpetion
             */
            manager.validateToken(token);

            /* 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);

            /* let us set some 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");
            /* let us add a listener to the SSOToken. Whenever a token
             * event arrives, ssoTokenChanged method of the listener will
             * get called.
             */
```

Code Example 4-8 SSOTokenSample.java File

```
            SSOTokenListener myListener = new SampleTokenListener();

            token.addSSOTokenListener(myListener);
        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Chapter 5

Understanding DSAME XMLs and DTDs

This chapter provides information on the two types of XMLs and DTD files used in DSAME, and how they can be used to create custom services. The two major types of XML and DTDs are:

- service XML files that are defined per the `sms.dtd` (Services Management Service) Document Type Definition
- batch update XML files are defined per the `amAdmin.dtd`. (These batch update XML files are used to perform creates, gets and deletes on various objects (users, groups, roles, people containers, organizations, etc.) in the Directory server tree

The service XMLs are used to define DSAME internal services and custom or external services. These are defined according to the rules defined in an XML DTD (called `sms.dtd`) in DSAME. Service XMLs and the service DTD is described in this chapter.

The other type of XML and DTD in DSAME is the batch updates XML files and DTD (amAdmin.dtd). These batch update XML files can be used by an administrator or customization engineer to perform batch updates to the Directory server DIT, such as create, get, or delete users, roles, groups, organizations, people containers, etc. This chapter provides some information on the batch update XML file, the `amAdmin.dtd` which defines rules by which to perform the batch updates to the DIT. Also refer to Chapter 6, “Using the Command Line Interface for information on using the amadmin CLI tool, which administrators and service developers must use to load the service XMLs into DSAME, and to load the batch update XML files.

This chapter provides the following sections:

- Understanding DSAME Services

Understanding DSAME Services

In DSAME, a service is a group of similar attributes defined under a common name. The common name is the service name. The attributes for a service can be attributes from an external service or application like Calendar or they can be from a internal service within DSAME such as authentication, session, logging, or URL policy agent. A service can also be a random set of attributes that a customer might want to group together and manage as a service.

A service can have attributes (user and service) and policy (privilege) attribute / value pairs which are described in service XML files. Attributes are simply configuration parameters with default values assigned to them. A service can contain policy attributes (which define privileges). After services are defined, they can be loaded (imported) into the Directory server using the `amadmin` CLI tool. Adding the service to DSAME enables an administrator to manage and configure the attribute values in iDS.

A service in DSAME is a set of attributes that are grouped together so that they can be manageable from the DSAME console. This service could be something like “Ford Car Parts”, into which you could add 30 attributes, and they could then be managed as a service within DSAME. The DSAME console lets you set and get these attributes; it does not interpret the behavior. Interpreting the behavior is performed by the client application that reads attributes from Directory server.

In DSAME, each service can be modelled as an Directory server auxiliary class with the attributes specified in the XML. Service metadata and default values taken from the XML are stored in a service config branch of the DIT: `ou=services,<rootsuffix>`, for example, `ou=services,o=iplanet.com`. (`<root_suffix>` represents the top of the Directory server tree, for example, `o=sun.com or ibm=sun.com or o=iplanet.com`.

All the service XML files in DSAME (whether internal or external services that customers might write and integrate) must adhere to the sms.dtd which is located in the `<dsame_root>/SUNWam/web-apps/services/dtd` directory. Service XML files describing each DSAME (internal) service can be found in `<dsame_root>/SUNWam/web-apps/services/WEB-INF/config/xml` directory.

Things to Consider about DSAME Services

As mentioned previously, a *service* in DSAME is a grouping of like *attributes* defined under a common name. The common name is the *service name*, for example, “bronze_engineering_mail_service”. The attributes for a service can be attributes from an external application like Calendar or they can be from one of the internal services within DSAME such as Authentication, Session, Logging, or URL Policy Agent.

A service can also be a random set of attributes that you might want to group together and manage as a service. For example, a customer might have an auxiliary object class with 10 attributes that are common to all user entries in their Directory server. To manage these attributes through DSAME, they must be described in a service within DSAME. One option would be to add them to the iPlanetAMUserService service. (You can add these attributes through DSAME

console or by modifying the amUser.xml file.) This is the service within DSAME that, by default, includes many common attributes from the inetOrgPerson and inetUser objectclasses. This service could be extended to include the attributes in the customer's object class, or the customer might choose to create a new service to manage these attributes. Whether you are extending an existing service or creating a new service, the attributes, after being imported or loaded with the `amadmin` CLI tool, would be manageable through the DSAME console.

It is important to understand that adding a service to DSAME enables DSAME to manage the attribute values in iPlanet Directory Server (iDS). It does not implement any behavior behind those attributes or dynamically generate any code to interpret the attributes. This means that DSAME can set and get the values, but it is up to an external application to interpret and use these values.

Continuing with the example above, if attributes common to all users were added, an administrator can then set those values or add them to roles or organizations through the DSAME console. For example, a Calendar or phonebook application might typically read those values from the Directory server. An administrator could add new users, setting those new attributes through the DSAME console, but the phone book application would be reading those attributes from the Directory server when querying for specific users.

Internal Services vs. External Services

Note that in the DSAME console under Services that there are many DSAME services shipped with the product—Logging, Naming, Session, URL Policy Agent, etc. These services are managed in the same way as external services. The difference is that DSAME provides code implementations that use these internal services' attributes.

A good example of this is the DSAME URL Policy Agent service. This service defines three attributes that are used by the web agents to check user access to URLs. The DSAME console allows the administrator to configure these attribute values, but the web agent is the external DSAME application that is using those attributes. The core DSAME features (such as logging, authentication, session services, etc.) are each described and managed as any external service would be managed. You can view the XML files that describe each DSAME service in the xml directory under `/SUNWam/web-apps/services/WEB-INF/config/xml`. This is good place to start when describing or adding a service to DSAME.

Service Schema defines service attributes and optionally default values

A service in a DSAME XML file contains a schema definition section and optionally default values defined within the schema section. Note that in DSAME, the internal services and the sampleMailService files provided in the sampleMailService directory in the DSAME product have only schema definitions sections, where default values are defined. (These defaults appear in DSAME console after the services are loaded into DSAME.)

The *schema* section of an XML file defines attributes (also called *configuration* parameters), and optionally default values for each attribute schema defined in the service schema section. The schema section of a service XML file is where a service developer or customization engineer defines configuration parameters for a service and some default values (optional). Note the global schema definition from the `sampleMailService.xml` file in Code Example 5-1 on page 89 which defines a global schema attribute and some default values.

Code Example 5-1 Global Schema definition with default values from sampleMailService.xml File

```
<ServicesConfiguration>
    <Service name="sampleMailService" version="1.0">
        <Schema
            i18nFileName="sampleMailService"
            i18nKey="iplanet-am-sample-mail-service-description">
    <Global>
            <AttributeSchema name="serviceObjectClasses"
                type="list"
                syntax="string"
                i18nKey="">
                <DefaultValues>
                    <Value>iplanet-am-sample-mail-service</Value>
                </DefaultValues>
            </AttributeSchema>
    </Global>
```

In the case of the `sampleMailService.xml` provided in the `<dsame_root>/samples/admin/cli/sampleMailService` directory and the other internal DSAME service XMLs provided in the DSAME product, the schema is defined along with any default values that the service developer wants to define for that service's attribute. This way, these default values display in DSAME console, and speed the process of creating service templates for administrators.

What Happens When you Register a Service

To register a service, the absolute requirement is that you must define schema in a service XML. Optionally, a service developer can define default values for the attributes within each attribute schema definition. (See Code Example 5-1 on page 89 for an example of an attribute schema definition with some default values defined.) If you do not define default values for the attributes defined in the schema, then no default values will display in the attribute fields for the service in DSAME console.

After registering a service (either through `amadmin` or DSAME console), the configuration data (which is the attribute schema definitions with any default values) is uploaded for that particular service. Then the administrator or customization engineer can change the configuration parameters' values from the default values, if they exist, through the DSAME console. The configuration data is what the administrator sees displayed in the DSAME console after the service XML schema and configuration data is imported using amadmin.

Overview of Services Management in DSAME

This section provides some information on the types of service management provided by DSAME. DSAME enables different types of administrators perform three types of management:

- *Services* and *User* management—The *Services Management* module (comprised of *services* and *user* management) provides a solution for registering services and managing service and user attributes. The Services Management service (module) is also comprised of user management, which deals with managing the structure of a customer's directory. This includes creating, deleting, and getting organizations, sub-organizations, groups, roles, and users.
- *Policy* management—The *Policy Service* provides solutions for managing (defining, modifying, and removing) policies for resources, and to determine users' privileges to resources. It deals with the creation of policies and how they are applied to a role or an organization.

In the iPlanet DSAME product, there are 17 services (including DSAME console, Logging, Session, six Authentication services (such as LDAP, Radius, Certificate, Anonymous, Self-registration (Membership)), URL Policy Agent, Naming, etc.) that get installed by default when the product is installed. (These DSAME services are sometimes referred to as "internal" services, as distinguished from "external", or customized services that customers may add to DSAME.) If customization engineers or service developers want to add new services that are external to

DSAME, they must write service XML files and load them into DSAME, for DSAME to be able to recognize and manage the service's attributes. Service developers must write service XML files when creating a custom service if you want a service's attribute / value pairs to be manageable through DSAME console. For example, if creating a custom pluggable authentication service in DSAME 5.0, you must create a new service XML for your custom service and load it into DSAME using the `amadmin` CLI tool. (See Chapter 6, "Using the Command Line Interface for information in using the `amadmin` CLI tool to load services.)

Services Management Module in DSAME

DSAME's service management component provides a mechanism for services to define and manage their configuration data. It is during the development stage that service schema and configuration parameters get defined (typically by a customization engineer or service developer), and it is during the deployment stage that these parameters get configured (typically by an deployment engineer or administrator). Consider, as an example, the implementation of a *mail* service. It is during its development that configuration parameters such as cache size, mail quota, and mail servers get defined, typically by a service developer who is developing a service or application. Then, during deployment, administrators and / or users typically configure these service attributes, also called "configuration parameters."

DSAME's service component provides a mechanism for service developers to define their configuration parameters and optionally provide default values. The mechanism is through multiple service XML files that must adhere to the DTD defined by DSAME (`sms.dtd`). The DTD and some sample service XML files are described later in this chapter. The definition of the configuration parameters and optional default values in the service XML file is called the "service schema" and sometimes referred to as "configuration data."

Services typically have different kinds of configuration parameters. For example, certain parameters are applicable to an entire DSAME installation (such as port number and server name), and some are applicable to users (such as mail quota). These different kinds of configuration parameters are stored and managed differently. The services management services component in DSAME provides the following service attribute categories for applying and differentiating control parameter definitions.

In DSAME, all the service attributes you define in service XML files must be one of the following five types of service attributes:

- global

- organization
- dynamic
- policy
- user

These service attribute types are described in more detail later in this chapter, along with examples of each type of service attributes, and when a service developer would use them when defining a custom service.

The following sections provide brief descriptions of the five attributes used by DSAME 5.0.

Global Attributes

The following apply for global attributes:

- **Global**: parameters that are common across a DSAME configuration for all services, and for the entire DSAME installation. Changes to these parameters would affect the entire service.

  Examples: A port number or a security algorithm that must be used by all servers. Cache size and maximum number of threads.

  Global parameters or attributes are only manageable from the DSAME console Services page. They cannot be configured for an organization, user, or role.
- Cannot be configured for organizations, roles, or users.
- They are global across a DSAME configuration.
- Values of these attributes can be modified using the Service Management page in DSAME console.

In DSAME 5.0, organization and global attribute data cannot be accessed from an external application. This is because these attributes are not stored as LDAP objects in the Directory server; they are stored as static blobs of data, thus they are not accessible by using LDAP commands. The global and organization attribute types can be considered private in DSAME 5.0, except for when creating custom authentication services.

Organization Attributes

- **Organization**: parameters that can be configured differently for various organizations. Changes to these parameters would affect only the organization. An organization attribute does not use CoS. In DSAME, all the internal authentication services are defined at the organization level.

  Example: Authentication services used by an organization.

  When defining a custom authentication service XML in DSAME 5.0, you use global and organization attributes only.

Dynamic Attributes

The following apply for dynamic attributes:

- These attributes can be assigned to roles or organizations which are then inherited by users that possess the role or belong to an organization.
- Default values can be specified.
- When service attributes are assigned to more than one role or organization, a conflict can arise. In such cases, there could be two possible behaviors, attribute aggregation (merge-schemes) or single value based on template priority (default).
- Uses CoS (class of service). (For details on CoS, see the Directory server documentation.) A dynamic attribute/value pair set at the organization level will be inherited by all subentries including users. For example, a dynamic attribute set in a role will be inherited by any user that possesses that role.
- Parameters that apply to all user objects. Changes made to these parameters would affect all the user objects.

  Examples: Organization's address; mail servers; company name; manager's name.

  Dynamic service attributes are modelled using CoS (Class of Service). A dynamic attribute set at the organization level will be inherited by all subentries including users. A dynamic attribute set in a role will be inherited by any user that possesses that role. (Dynamic attributes function and are used internally the same as policy attributes.)

Policy Attributes

The following apply to policy attributes:

- Policy attributes are a special type of dynamic attribute. Policy attributes function identically to dynamic attributes in DSAME. Policy attributes are used specifically only when defining policy objects.
- Used to create named policies under an organization.
- There is no support for default values.
- Currently, DSAME has only one service with policy attributes—URL Policy Agent.
- Attributes for URL Policy Agent use the CoSQualifier aggregation option. The default is aggregation. (CoSQualifier has two settings: aggregation (default) or merge-schemes.)
- Policy (and dynamic) attributes can be retrieved and accessed using LDAP commands; they are stored as LDAP objects in the Directory server.
- Policy attributes are parameters that define the privileges provided by the service.

  Example: A web server defining the URL access privileges through either GET or POST.

Administrators can manage policy attributes only from the Policy page in DSAME console. You cannot set policy attributes at the user level—only the organization and role level.

Policy or named policy objects are not manageable by LDAP; they are stored as static blobs of data in Directory server.

User Attributes

The following apply to user attributes:

- Attributes that will only be part of the user entry, and that different for each user entry.

- User attributes are parameters that are specific to a user. Changes to these parameters would affect a particular user. Configuration information that can only be set in a specific user entry; it cannot be set at the role and organization level.

  Example: A user's locale. Another example of a user attribute would be password.

  Administrators can only set user attributes in a user entry—not in a role or an organization entry.

Although the service management module provides support for storing and managing the schema for the service, it manages schema data only for *global* and *organization* types. The *user* and *dynamic* configuration data is managed by the user management module, and *policy* configuration is managed by the policy management module. However, both user management and policy rely on service management for service schema management. The elements and attributes in the DTD (`sms.dtd`) that are supported in DSAME 5.0 are described later in this chapter. Service developers can use the supported elements and attributes to described later in this chapter in their custom services.

The service schema and any default values defined within the schema section in a service XML file are stored in the Directory Server. (Note, however, that global and organization attribute data is stored as static blobs of data, thus global and organization attribute data is not accessible through LDAP commands. Also, they should be used only when writing custom authentication services in DSAME 5.0.

Defining and Adding Services to DSAME

This section provides some information to consider when using existing applications whose attributes you want to be manageable from DSAME console, as well as when creating new services to be manageable by DSAME console.

Adding a Custom Service to DSAME

If you are writing an application or if you have an existing application that already has attributes in the Directory server, and you want that application's attribute data to be manageable from DSAME console, then you need to define a service XML using the elements and attributes that DSAME 5.0 supports. (See the sections in this chapter that describe the supported elements and attributes for defining services in DSAME 5.0.)

If an application already uses certain Directory server attributes, DSAME should be able to manage these attributes with little customization required. You might need to change some of the attributes so that the application can get certain attribute values from the user.

Typically, you could model user and dynamic attributes and the application would work and you could manage the attributes through DSAME console. For example, for an existing customer application “BronzeMailService1”, a customization engineer could create a service XML file for that mail service called “BronzeMailService1.xml”, define some attributes in it, then run the `amadmin` tool to load the service and its attribute/value pairs into DSAME, modify certain configuration files, and then the attributes will be manageable from the DSAME console, and will be readable and writable by Directory server.

NOTE There can be only one default value for a particular Global attribute (configuration parameter) for the entire platform. These attribute values are defined in the service XML files with “global” schema attribute types.

When a service developer or customization engineer is configuring DSAME, he/she configures attribute schema and default values within the schema section in the service XML files. Later, when an administrator enables a service for an organization in DSAME console, the attribute schema and default values that were specified previously in the service XML file will display, thus be manageable, in DSAME console. For example, when an administrator enables Authentication under the Services page, you would see the runtime defaults that were configured at installation time when DSAME loaded in the service schema for all the attributes for that service, along with default values for the attributes, from all the service XML files.

When You Create a Service XML, Attributes Must be Defined (Default Values are Optional, but Recommended)

When the service is first created, however, its attributes and default values are defined in the XML files. Service XML files are loaded into DSAME through the `amadmin` CLI tool. When an administrator changes the default values in DSAME console, he/she can change those attribute values, then click Submit to create a modified service template for an organization or a role.

Remember that you only need to define a service XML if you want that service’s attributes to be manageable from DSAME console; otherwise, there is no need to define a service XML.

You must model any new services in service XML files if you want that service's attributes to be manageable from DSAME console.

Attribute value in schema provides a default value for administrators and users

The attribute value a service developer defines in a schema section of a service XML file provides a default value for that attribute or configuration parameter. The administrator can always change the default value to some other value in DSAME console.

After importing a service schema with attribute / value pairs defined using `amadmin`, then the administrator can change the values from the defaults on an as-needed basis, then click Submit to create a new template, which creates a new configuration template.

DSAME's service XML files provide default values for service attributes

DSAME has the notion of default values for service attributes. When describing an attribute for a service in XML, the default value(s) can be set. The defaults specified in the service XML files are used when adding the service attributes to a role or organization. At the time an administrator configures the attributes for a role or organization in DSAME console, a page is displayed with the default values. Administrators can change the defaults or the submit the service template using the defaults. Or, the default values can be set in the service management pages in DSAME console. These defaults are global and can not be set per organization.

Defining an empty attribute value in Schema

A service developer could also define an empty field in the schema definition with double quotes. For example, you could define " " as one of the attribute values.

Code Example 5-2 Excerpt showing an empty attribute value in schema

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

Using DSAME to manage attributes in your existing DIT

You may want to use DSAME to manage attributes in your existing DIT. Your existing DIT might happen to have a lot of customized user attributes. In this case, you could define all your attributes to be User attribute types in either a User service XML file (defined in the `amUser.xml`) or a service XML file (which could be a customized service XML file you add to DSAME, called an "external" service XML file).

When Adding a new Service or Application, You must Define Schema (Object classes and attributes) in Directory Server

If a service or application's attributes do not already exist in the Directory server's schema, you need to define the schema (object classes and attributes) in Directory server. You would need to add object classes for every new attribute that the Directory server does not already have in its schema so that DSAME and Directory server can manage those attributes.

One way to define new service attributes in DSAME

A customer could also define new service attributes. To add attributes to the Directory server schema, one way is to go to Configuration / Schema page in Directory Server console which contains the schema for the Directory server.

Adding an Existing Application to DSAME

Typically, if a customer is using an existing application, and wants its attributes to be manageable by DSAME, the schema is probably already defined and has been loaded into Directory server, so that Directory server already knows what attributes and object classes that application uses.

If Directory server does not already have your existing application's attributes and object classes loaded into it, then you need to update the Directory server with the new or modified objectclasses and attributes using the ldapmodify command or by using Directory Server Console.

Enabling a Service for an Organization or Role in DSAME console

In the DSAME console, you assign service attributes to an organization or a role by enabling that service for the organization or role, then creating a service template and modifying the attributes for that service. After you have created the template and set the values, all users under that organization will inherit those service attributes.

NOTE For every node or object underneath that organization or that role will inherit that service's attribute / value pairs only if dynamic attributes have been assigned at the organization level or the role level.)

To assign the service attributes to a role, you must first register the service in the role's parent organization. Then the administrator could configure the service for the role and set the values. When a user is assigned to the role, he or she will inherit the attributes from that role, provided that the attributes defined at the organization or role are defined as dynamic attributes. A user can have multiple roles thus would inherit attribute values from multiple parent organizations.

Defining Global Attribute Types in a Service

You could also define or have attributes in a service that are "global". For example, an application might use a global attribute such as "port number", and it also might have an attribute such as User phone number, which is always specific to a user. A User attribute is one that would not make sense for it to be applied to the entire organization, for example, but would apply only to a specific user, for example, FirstName and Last Name, or EmployeePhoneNumber. The DSAME console shows a User attribute only in the context of a specific user; it does not apply to the organization or role level.

Global attributes are attributes that cannot be applied to organizations, roles or users. They are global across a DSAME configuration. They are typically used for things like ports, protocols, and server names. Global attributes are settable in the DSAME console under Service Management.

NOTE In DSAME 5.0, attributes defined as Global (and Organization) can only be accessed using the DSAME SDK. Global and organization attributes are stored as static blobs of data in the Directory server, and are not accessible by using LDAP commands such as `ldapsearch`.

Because the DSAME SDK is not public (not exposed to customers for customization purposes) in DSAME 5.0, when customizing or creating external services, you should not use the Global or Organization attribute type (except for Authentication services). The global (and organization) attributes are the only attributes that you should use for your custom authentication services in DSAME 5.0.

Global attributes are platform-wide attributes that are global across an entire configuration, for example, server_host or port_number. The values of these attributes can be modified through the Service Management page in DSAME console.

Global attributes are typically applicable to things such as protocol, host name, server name, and port number. Global applies to attributes that apply across the entire platform or DSAME installation. Global attributes are attributes that everyone using that platform configuration will get by default. After an administrator configures a service for a particular organization or role in DSAME console, he/she could customize that service and its attributes thus creating a new service template, and then assign that service template to an organization or a role.

Typically, authentication modules in DSAME use only global and organization attributes.)

The Services page in DSAME console displays the default attribute/value pairs for each service that has been previously loaded into DSAME. For example, global attributes can be managed by administrators through the Services page. A global attribute might be `platform locale`. Whenever DSAME performs logging, it logs to the platform locale; it should be logged on one platform locale. The parameters `loginUrl and logoutUrl` also apply to the entire platform, thus are defined as global attributes. Naming service attributes are typically also defined as global attribute types.

Global attributes cannot be applied to organizations, roles, or users.

DSAME allows you to manage and change attribute values, either through XML files or by modifying default values for attributes, then creating a new service template.

When you display a role or an organization for a service, you get a generic Edit page. All that Edit page does is grab all the attribute values and types from the service XML and displays them; it does not interpret the attribute/value pairs and behavior.

Defining Organization Attributes in a Service

Authentication attributes are a good example of Organization attributes. Because authentication would apply on a per-organization basis, it would not make sense for the attributes to be dynamic since all users would inherit those attributes. All Authentication service attributes in DSAME are defined as Organization type.

Organization attributes are attributes that only apply to organization entries and when it is not desired that users inherit those attributes, as would happen if they were defined as dynamic attributes.

NOTE In DSAME 5.0, attributes defined as Organization (and Global) attributes can only be accessed using the DSAME SDK. Global and organization attributes are stored as static blobs of data in the Directory server, and are not accessible by using LDAP commands such as `ldapsearch`.

Because the DSAME SDK is not public (not exposed to customers for customization purposes) in DSAME 5.0, when customizing or creating external services, you should not use the Global or Organization attribute type (except for Authentication services). The only service in DSAME 5.0 that global (and organization) attributes can be used for are when creating a custom pluggable authentication service.

However, global and organization attributes are used by the DSAME internal services, such as session, naming, logging, authentication, etc.

Defining Dynamic Attributes in a Service

Dynamic attributes are attributes that can be assigned to roles or organizations which are then inherited by users that possess the role or belong to the organization. Dynamic attributes promote ease of administration and scalability by allowing attributes to be assigned to roles or organizations instead of setting these attributes in each user's entry. (See Section "Adding attributes Common to all Users to the User Service in DSAME" on page 125.) When these attributes change, the administrators only have to change them in the role or organization instead of in every user entry. Also, by setting the attributes in a top level or suborganization they can be used as defaults unless overridden by roles or suborganization attributes that have a higher priority.

An example of a Dynamic attribute might be a mail server. Typically, an entire building might have a common mail server, so each user would have a mail server attribute in their entry. If the mail server ever changed, every user attribute would have to be updated. If the attribute was in a role that each user in the building possessed, then only the attribute in the role would have to be modified. That, of course, is a very simple example, but a role or organization may have many attributes from different services assigned to it.

Dynamic attributes are attributes that are modelled using class of service (CoS in Directory server). (Dynamic attributes in DSAME use the iPlanet Directory Server CoS and roles feature. For more information on how CoS works in Directory server, refer to the Directory server 5.0 documentation.) A dynamic attribute that is set at the organization level in DSAME console will be inherited by all subentries including users. A dynamic attribute set in a role will be inherited by any user that possesses that role.

Organizations and Dynamic Attributes

The following things apply to organizations and dynamic attributes:

- Multiple services can be registered per organization or sub-organization.
- Attribute and policy values of an activated service are customizable per organization.
- After service activation, all the entries in the sub-tree of the organization inherit the service attributes.

Defining Dynamic and Policy Attributes

Because DSAME uses CoS, the service definitions typically need to be defined at the organization level. So, if an entry has a certain CoS entry, it gets a CoS definition. Roles know that based on this filter, all these users have this role. It is done differently for regular roles for users; they just have a pointer to that role.

Dynamic attributes can be defined for an organization, for an organizational unit (ou), and they can be defined for a role. It depends on how people want to manage their users. You do not even have to use roles at all; you could use organizational unit (ou's) instead.

In the example discussed previously, where you have a manager and an employee, it would be better to have one role that is for a manager and an employee, instead of one role for a manager and one role for an employee and then aggregate it.

Or another way is just to aggregate in all the other attributes. What DSAME does is it performs a get operation on the User object, and then based on this schema data, it can tell what service it's from. DSAME can distinguish among different types of attributes, for example, it can determine which are your authentication attributes, which are your session attributes, etc.

Roles in DSAME

Roles in DSAME don’t have anything to do with ACIs, at least within the context of looking at roles from the DSAME console. A role is just a way to group some attributes for services which you can then apply to a user, based on whether that user possesses that role or not. Internally, DSAME uses roles for administrator permissions based on ACIs. Roles in DSAME are just a group of service-related attributes, which are viewable from the DSAME console.

In DSAME, a role uses the `nsrole` attribute. An administrator can add or assign services and attributes to roles. In Directory server, roles are primarily a way to group users. However, in Directory server, it is recommended that if you are creating a new DIT, you should use roles instead of groups because they are more efficient. In Directory server, a role is an entry grouping mechanism that allows applications to locate the role of an entry, rather than select a group and browse the members list.

For example, for inheritance to work, you could define a role that is located within an organization that has a role filter of objectclass=*. What that means is that it's a filtered role which means that if you pass this filter, the entry's got that role. Thus, objectclasses with an asterisk at this level means that every subentry has that role.

What Happens when a User is Assigned to a Role

When a user is assigned to a role, that user inherits the attributes from that role. A user can have multiple roles and inherit attribute values from multiple parent organizations. DSAME has the notion of default values for service attributes. When describing an attribute for a service in XML, the default value can be set. The defaults are used when adding the service attributes to a role or organization. At the time an administrator configures the attributes for a role or organization, a page displays with the default values. The administrator can change these defaults and submit a newly-created template, or submit the template using the defaults. The default values can be set in the service management pages. The defaults are global (that is, they apply to the entire DSAME installation or platform, and may not be set per organization.)

Overview of Roles in DSAME

This section provides some information on how DSAME uses roles, ACIs, and CoS (Class of Service).

In the DSAME 5.0 release, there is no notion of an “administrator” or “admin” in the DSAME console; that is, administrator roles or admin roles are not distinguished from other roles.

The admin roles are not differentiated from other roles that might be created by administrators in DSAME console. For example, when you are at an organization level in the DSAME console, for example, o=sun.com, and you go to the Roles page, you can view all the roles. These roles could be administrator roles, such as HelpDeskAdminRole, or a Policy_role, a mail_role, an OrgAdminRole, etc. All the roles are listed on the same page. If you have 100 roles, you see a list of 100 roles.

When you sit at the organization level, you get an entire list of roles. Say, you wanted to find the admin for PeopleContainer, which may be down another level. You have to go to the organization level, then look for the role based on the name, click on that, then add the user to the role.

When you are at the Organization level, and click on suborganization, you see all the roles—administrator-type roles (TopLevelHelpDeskAdmin Role, HelpDeskAdminRole, OrgAdmin, OrgHelpDeskAdmin, etc.) and other types of roles created by administrators. These could be termed "access control roles", as they are roles that define a user’s access. There could also be some "service roles", for example, "mail_service_role" or "authentication_service_role".

You can no longer go to the PeopleContainer, for example, and view your roles there. You must go up the Organization level and then view the roles there.

The notion of a role represents access control. The point of a role is to get service attributes and policy attributes. Within DSAME console, an administrator can assign a service to a role.

How DSAME uses Roles

This section gives some information on how DSAME uses roles and CoS.

- Multiple roles can be created for each organization or sub-organization.
- A role can be enabled with any number of services with customized attributes and policy.
- By adding a service to a role, any user with that role will inherit the service attributes and policy from that role. Users can have any number of roles.

DSAME uses Classic CoS and role templates. (For information on Classic CoS, see the iDS 5.0 documentation.)

Roles

Roles are a new entry grouping mechanism in iDS 5.0 that unify the static and dynamic groups. Roles are designed to be more efficient and easier to use for applications. For example, an application can locate the role of an entry, rather than select a group and browse the members list.

CoS

Class of Service (CoS) allows you to share attributes between entries in a way that is transparent to applications. CoS simplifies entry management and reduces storage requirements.

CoSQualifiers used by DSAME

- DSAME supports *default* and *merge-schemes* CoS qualifiers. The qualifier can be specified in the XML (service schema description) file when describing the manageable attribute of the service.
- If no CoS qualifier is specified, *default* is assumed.

As an example, the URL Policy Agent uses *merge-schemes* to obtain aggregated values for Allow, Deny, and Not Enforced attributes.

Organizations and CoS

- Multiple services can be registered per organization or sub-organization.
- Attribute and policy values of an activated service are customizable per organization.
- After service activation, all the entries in the sub-tree of the organization inherit the service attributes.

Registering a Service Creates a CoS Definition and CoS Template

Registering a service for a particular organization creates a COS definition for that service.

A class of service (COS) allows you to share attributes between entries in a way that is transparent to applications. COS simplifies entry management and reduces storage requirements.

There are three COS definitions:

- Indirect COS Definition
- Direct / Classic COS Definition.

- Pointer COS Definition.

When an imported service gets registered for an organization, a Direct / Classic COS Definition is created for that service based on the policy template type and non-policy template type.

A classic COS identifies the template entry by both its DN and the value of one of the target entry's attributes.

This service must have been imported by using the amadmin CLI tool for that particular organization.

Roles in DSAME are at a higher level of abstraction than Directory server roles

Roles are a new entry grouping mechanism in Directory Server 5.0 that unify the static and dynamic groups. Roles are designed to be more efficient and easier to use for applications. For example, an application can locate the role of an entry, rather than select a group and browse the members list.

Roles in DSAME do not function in the same way as Directory server roles. They are a higher level of abstraction. DSAME roles are an abstraction of Directory server roles.

Access control in DSAME is implemented using Directory Server roles. Users in DSAME inherit the directory access permissions using directory roles. ACIs are not created using the `groupdn` as the subject.

When DSAME is installed, eight different administrator role types are created. Administrators can create new admin roles and modify them through DSAME console.

Default ACIs for the roles are stored in a global configuration attribute in the `iplanet-am-admin-console-default-acis` attribute. This attribute is defined in the iPlanetAMAdminConsole service XML profile. Default ACIs can be modified through the DSAME console.

How Organizations and Roles use Dynamic Attributes

Organizations and roles in DSAME frequently use dynamic attributes. Dynamic attributes would typically be used when you want to define a role at the organization level and have all the users inherit that role and its attribute / value pairs.

One way to Add new attributes to user entries

If an administrator does not care about defining a role at the organization level and having all the users inherit a particular attribute and its values, then the customization engineer could define all the attribute types as *User*. For example, you might have 50 auxiliary classes that all users possess. To manage those 50 auxiliary classes, you might define one service XML file that defines all these auxiliary classes, then load them into DSAME using `amadmin`. Those attributes would then appear in the DSAME console.

If you have an existing DIT, you may not care about using dynamic attributes. You may not care if it's time-consuming to change one attribute for 50 users. One way a customization engineer might make these changes is to just modify or add the attributes for the 50 users through Directory Server Console.

How Dynamic Attributes are used in Roles

For example, you might decide to create a contractors role which has attributes from the session service and the URL Policy Agent service. Then, when new contractors start, the administrator could just assign them a single role instead of setting all the attributes in the contractors entry. If the contractor were to become a full-time employee, the administrator could just assign them a different role.

How Dynamic Attributes are used in Organizations and Roles

In the DSAME console, you assign service attributes to an organization by enabling that service for the organization, then creating a service template and modifying the attributes for that service, then assigning the service to an organization or a role. (Creating a template typically consists of changing some default values for a service in DSAME console, then clicking Submit to create the template. You can also create a template without changing any of the default values.) After you have created the template and set the values, all users under that organization will inherit those service attributes (only for dynamic attributes that are assigned to an organization).

How DSAME Dynamic and Policy Attributes Use CoS

Both dynamic and policy attributes uses CoS. Class of service (CoS) allows you to share attributes among entries in a way that is transparent to applications. CoS simplifies entry management and reduces storage requirements. For specific information on how CoS works, see the Directory server 5.0 documentation on advanced entry management.

Clients of the Directory server read the attributes on a user's entry. With CoS, some attribute values may not be stored with the entry itself. Instead, they are generated by Class of Service logic as the entry is sent to the client. Each dynamic (and policy) entry is comprised of the following two entries in your Directory server:

- CoS Definition Entry—the CoS definition entry identifies the type of CoS you are using.
- Template Entry—The template entry contains a list of the shared attribute values. Changes to the template entry attribute values are automatically applied to all the entries sharing the attribute.

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.

Dynamic attributes in DSAME work in such a way that any subentry under o=sun, and all the way down the tree, if that entry is gotten directly (for example, if you do an LDAP search against the entry), you get back the attributes for that entry plus anything that is in the CoS template. This means that all the users are going to automatically get these values. For example, you might have your locale set by default. The top of your tree might be o=sun, and you might have two regions: east and west. For example, you could set the division attribute so that all users would get it. That way, if you move a user around, they automatically get the new attributes. With dynamic attributes (and for policy attributes also), you do not have to modify that user's entry. Or if you create a new user, you don't have to set those attributes; they automatically get those attributes by inheriting them from the organization level (because the attribute has been defined as a dynamic attribute, and assigned to that organization). It is also possible for users to inherit attributes from the role level.

Also, *dynamic* attributes save space in the DIT. For example, if you have 2 million users and you have 70 attributes, you would have to store that entry 70 times 2 million users. By putting this entry into a dynamic attribute, the Directory server only needs to take up the size of one entry. This typically applies at the organization or role level.

There are two important notes about *dynamic* (inheritable) attributes. They work at both the role and organization levels (that is, also at sub-organizations and organizational unit levels). In DSAME, organizations or sub-organizations (organizationalUnits) are viewed and handled similarly.

Roles and Dynamic Attributes

The following things apply to roles and dynamic attributes:

- Multiple managed roles can be created for each organization or sub-organization.
- A managed role can be enabled with any number of services with customized attributes and policy.
- By adding a service to a role, any user with that role will inherit the service attributes and policy from that role.User can have any number of roles.

DSAME uses Classic CoS and role templates.

Conflicts with multiple organizations or roles

There is the possibility that a dynamic or policy attribute could be assigned to more than one role or organization, thus creating a possible conflict. When this happens, there are two possible behaviors—*attribute aggregation* or *single value* based on template priority. When the attribute is described in the service XML, the CoSqualifier attribute can be set to one of the following:

- default
- merge-schemes

  default—If CoSqualifier is set to "default" or is missing, then the template with the highest priority is returned. When a template is created for an organization or role in DSAME console, a priority can be set for the template.

  merge-schemes—tells Directory server that when the attribute appears in more than one organization or role, the values should all be aggregated and returned as a multi-valued attribute.

Conflicts and Dynamic (or Policy) Attributes

DSAME supports the *CoS qualifiers* "default" and "merge-schemes" settings. The qualifier can be specified in the service schema definition in the XML file when describing the attribute for the service that you want to be manageable from DSAME console.

If no CoS qualifier is specified, *default* is assumed.

DSAME out of the box has only one service with policy attributes—the URL Policy Agent. For example, the URL Policy Agent service in DSAME uses the CosQualifier merge-schemes setting to obtain aggregated values for the Allow, Deny and Not Enforced attributes. This means that all policies from multiple roles or organizations will be aggregated. For example, if the "Employee Role" says that you can access

```
*/employee.html
```

and the "HR Role" says you can access

```
*/hr.html
```

then when my access is returned to the URL Policy Agent, it will get both

```
*/employee.hml
*/hr.html
```

If the URL Policy Agent attributes were defined as priority, you would get the list of URLs from the named policy with the higher priority. See the advanced entry management sections in the Directory server 5.0 documentation for detailed information on CoS.

Roles

This section provides some information on how DSAME uses the Directory server 5.0 Roles and CoS features.

DSAME provides for the dynamic configuration and management of users, services and policy. The basic building blocks of DSAME are organizations, groups, roles, CoS, services, and users.

A *service* is comprised of attributes and policy (privilege attributes) described in XML and loaded into the Directory server using the `amadmin` tool. Service attributes can be tagged as either `global, organization, dynamic, policy,` or `user`. Global attributes are service configuration attributes that are only modifiable on a platform-wide basis. User attributes can only be set for a particular specified user. Dynamic and policy attributes can be set at the organization, sub-organization, and role levels and are inheritable by sub-entries based on the iDS CoS feature. Organization attributes can be set for a particular organization, for example, mail servers used by an organization, or an authentication service (such as LDAP or RADIUS). Each service is modeled as an auxiliary class with the attributes specified in the XML. Service metadata and default values taken from the XML are stored in a service config branch of the DIT—ou=services,<rootsuffix>; for example, ou=services,o=iplanet.com.

Organizations and CoS

Any number of services can be enabled for an organization or sub-organization. When a service is enabled, the attribute and policy values can then be customized specifically for that organization. By enabling a service for an organization, all entries in the sub-tree will inherit the service attributes (provided they are defined as “dynamic” attributes) unless overridden with a lower CoS priority. To

accomplish this, DSAME uses classic CoS and role templates. There is a default filtered role with nsRoleFilter set to (objectClass=*) for each organization. DSAME uses the CoS template of that default filtered role to define default service attributes for all entries in the organization's sub tree.

The following is an example of an organization, its default CoS role and one service enabled.

Code Example 5-3 Organization Example with default CoS Role and One Service Enabled

```
dn: o=iplanet.com,o=isp
objectClass: top
objectClass: organization
dn: cn=ContainerDefaultTemplateRole,o=iplanet.com,o=isp
objectClass: top
objectClass: nscomplexroledefinition
objectClass: nsfilteredroledefinition
objectClass: nsroledefinition
objectClass: ldapsubentry
nsRoleFilter: (objectclass=*)
cn: ContainerDefaultTemplateRole

dn: cn=iPlanetAMAuthService,o=iplanet.com,o=isp
CoSspecifier: nsrole
objectClass: CoSclassicdefinition
objectClass: top
objectClass: CoSsuperdefinition
objectClass: ldapsubentry
CoStemplatedn: cn=iPlanetAMAuthService,o=iplanet.com,o=isp
CoSAttribute: iplanet-am-auth-menu
CoSAttribute: iplanet-am-auth-profile-required
CoSAttribute: iplanet-am-auth-admin-module
CoSAttribute: iplanet-am-auth-authenticators
CoSAttribute: iplanet-am-auth-login-workers

dn:
cn="cn=ContainerDefaultTemplateRole,o=iplanet.com,o=isp",cn=iPlanetAMAuthServi
ce,o=iplanet.com,o=isp
cn: "cn=ContainerDefaultTemplateRole,o=iplanet.com,o=isp"
cn: cn=ContainerDefaultTemplateRole,o=iplanet.com,o=isp
objectClass: CoStemplate
objectClass: top
objectClass: extensibleObject
objectClass: ldapsubentry
iplanet-am-auth-menu: LDAP
iplanet-am-auth-profile-required: false
iplanet-am-auth-authenticators: com.iplanet.dpro.auth.module.ldap.LDAP
iplanet-am-auth-login-workers: com.iplanet.dpro.auth.server.HTMLLoginWorker
```

Roles and CoS

Any number of managed roles may be created for each organization or sub organization. A managed role may be enabled with any number of services with customized attributes and policy. By adding a service to a role any user with that role will inherit the service attributes and policy from that role. Users may have any number of roles. To accomplish this DSAME uses classic CoS and role templates. The following is an example of a role with the name eng, the authentication service enabled and customized for the eng role, and a user with the eng role.

Code Example 5-4 Example of Role "Eng" and User with the "Eng" Role

```
dn: cn=eng,o=iplanet.com,o=isp
objectClass: nsmanagedroledefinition
objectClass: nssimpleroledefinition
objectClass: top
objectClass: nsroledefinition
objectClass: ldapsubentry

dn: cn="cn=eng,o=iplanet.com,o=isp",cn=iPlanetAMAuthService,o=iplanet.com,o=is
objectClass: CoStemplate
objectClass: top
objectClass: extensibleObject
CoSPriority: 10
preferredLocale: en_US
preferredTimezone: pacific

dn: uid=jamie,ou=People,o=iplanet.com,o=isp
cn: jamie nelson
sn: jamie
objectClass: iplanetPreferences
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetUser
objectClass: inetOrgPerson
nsRoleDN: cn=eng,o=iplanet.com,o=isp
```

Access to Directory server is through proxy authentication

All access to the directory is through proxy authentication. A proxy user is created at installation time and all access is through that user proxying as the requesting user.

Roles and ACIs

Note the following with regard to how roles and ACIs are used by DSAME:

- Access control for DSAME is implemented using iDS roles.
- Users in DSAME inherit the Directory access permissions using directory roles. ACIs are not created using the `groupdn` as the subject.
- DSAME, when installed, creates eight (8) different admin role types.
- New Admin roles can be dynamically created and modified using DSAME console.
- Default ACIs for the roles are stored in a global configuration attribute in the `iplanet-am-admin-console-default-acis` attribute. This attribute is defined in the *iPlanetAMAdminConsole* service XML profile (`amAdminConsole.xml`).
- Default ACIs can be modified.

Access control for DSAME is implemented using iDS roles. Before Directory server 5.x, access control would have typically used a similar design, but with groups instead of roles. Users would inherit directory access permissions based on group membership. Users in DSAME inherit the same types of directory access permissions using Directory server roles. DSAME does not create any ACIs using the `groupdn` as the subject.

When the iPlanet DSAME product is installed, nine administrator role types are created. Those types are Top Level Admin, Top Level Help Desk Admin, Org Admin, Org Help Desk Admin, OrgUnit Admin, OrgUnit Help Desk Admin, Group Admin, Group Help Desk Admin, and People Container Admin. These role types have a set of default ACIs that get set when each role is created by the DSAME SDK. The default ACIs are stored in a global configuration attribute—the `iplanet-am-admin-console-default-acis` attribute. This attribute is defined in the iPlanetAMAdminConsole service XML profile, and can be configured by an administrator or customization engineer by using the `amadmin` tool, `ldapmodify` or through the DSAME console service pages.

Because the default behavior of the product is to dynamically create the above administrator roles when the corresponding entry type is created, the DSAME SDK depends on these default role permissions to be present when creating the entries. For example, when an organization is created, the DSAME SDK creates an Org Admin role and an Org Help Desk Admin role. After creating these roles, the SDK gets the default permissions (ACIs) from the

iplanet-am-admin-console-dynamic-aci-list attribute and sets them in the specified entries. This allows customers to modify the default permissions for the dynamic roles. It is important to note that these default permissions are not types, they are, in fact, just defaults.

After the role is created, the ACIs for that role are stored in the role itself, so changing the default permissions in the iplanet-am-admin-console-dynamic-aci-list attribute after the role is created will not affect that role, but only affect roles created after the modification has been made.

If a modification to any role permission is desired, you must modify the iplanet-am-role-aci-list attribute for the role through the DSAME console or through Directory server Console. (The iplanet-am-role-aci-list attribute holds all the ACIs in string format for that role.)

The iplanet-am-admin-console-role-default-acis attribute are the permissions (ACIs) displayed in the role's Create window of DSAME console. If you want to add more default permissions, add them to the iplanet-am-admin-console-role-default-acis attribute.

When a role is created in DSAME, it will have the auxiliary class iplanet-am-managed-role with the following attribute:

```
iplanet-am-role-type
```

This will be one of the following three values:

- 1 for top level admin
- 2 for general admin
- 3 for user

This attribute is used only by the DSAME console for display purposes. After authentication, the DSAME console gets the user's roles and checks the types of each role. If a user has no administrator roles, the User profile page will displays. If they have any admin type of roles, the DSAME console will make an DPStoreConnection.getTopLevelContainers() SDK call and start the user at the top-most view in the entry hierarchy. There is a catchall attribute iplanet-am-user-start-dn that can be set to override the starting point for the administrator. This attribute is needed because an administrator's desired starting point in the DSAME console user interface might be something other than what is dictated by the ACIs. This attribute can be set in conjunction with a role.

For example, if a group administrator happens to have read access to the top level organization, he or she might still want their starting point in the DSAME console to be the group they are managing. If the console went by the entries returned by the SDK based on ACIs, the admin would start at the top-level organization.

`iplanet-am-role-description`—The text description for the role. An administrator or user enters the description is entered through the user interface when creating a new role. The description for the dynamically-created roles is based on the names in the localization files.

`iplanet-am-role-aci-description`—The text description for the access permissions. The intent of this description is to describe what the ACIs for this role provide in an easy to understand format. This attribute can be set when creating the role. There is also a default description for the default ACIs that is used when roles are dynamically created or when a predefined permission is selected in the role creation pages of the DSAME console.

`iplanet-am-role-aci-list`—The list of ACIs that will be set when a role with this permission name is created through the DSAME console or by the SDK. The format of this attribute is the following:

permissionName | ACI Description | DN:ACI ## DN:ACI ## DN:ACI

permissionName—The name of the default permission

ACI Description—The readable (by people) description of the access these ACIs allow. The creator of these entries should assume that the reader of this description does not understand ACIs or even Directory server concepts.

DN:ACI—There may be any number of DN:ACI separated by the "##". The SDK will set each ACI in the DN entry. This format also supports tags which will 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 you have a default set of ACIs for every Org Admin, you would not want to hardcode the organization name in the default ACIs for the Org Admin. The supported tags are ROLENAME, ORGANIZATION, GROUPNAME, and PCNAME. These tags will be substituted with the DN of the entry in the SDK create method when the corresponding entry type is created.

If there are duplicate ACIs within the default permissions, the SDK will not fail, but it will print a debug message (not an error message). This will be a common occurrence when macro ACIs are used.

Defining Policy Attributes in a Service

Policy attributes are a special type of *dynamic* attribute. They behave and follow the same basic design as Dynamic attributes, and function internally the same as dynamic attributes.

Describing an attribute as policy allows the DSAME console to distinguish between policy and dynamic attributes. The main difference between them is that policy attributes from one service can be used to create named policies under an organization. The named policies can then be assigned to roles or organizations.

With dynamic attributes, you can directly assign attributes from a service to an organization or role. There are no named service objects. In the DSAME console, there is a separate pulldown for policy management. This is where the organization administrator can define the named policy objects which can be later assigned to organizations and roles under the user management pulldown pages. For example, an organization administrator might define the following named policies: "Admin URL Access", "HR URL Access", "Contractor URL Access", and "Employee Access".

In the user management pages, these named policies can be assigned to roles or organizations. The administrator might assign the "Employee Access" policy to the organization so that all employees in the organization inherit the policy. Then the administrator might assign the "HR URL Access" policy to the "HR Role". This way, all users in HR will get those policies along with any policies from the organization level.

There are no default values supported for named policy objects. This means that administrators must create named policies from scratch in the DSAME console. When a new named policy is created,there are no defaults as in the service management pages.

Policy Attributes

Policy attributes provide a way to control access to resources, and to determine users' privileges to resources.

NOTE The policy component in DSAME uses the configuration definition section in a service XML, but it creates the configurations through SMS APIs, not through XML files.

Administrators can assign policies to organizations or roles in DSAME console

In the DSAME console there is a separate pull-down for policy management. This is where the organization administrator defines the named policy objects which can be assigned later to organizations and roles under the User Management pulldown pages. For example, an organization administrator might define named policies called "Admin URL Access", "HR URL Access", "Contractor URL Access", and "Employee Access". In the User Management pages, an administrator could assign these named policies to roles or organizations. The administrator might do the following:

1. Assign the "Employee Access" policy to the organization so that all employees in that organization inherit the policy.
2. Assign the "HR URL Access" to the "HR Role".

   This way all users in HR will get those policies along with any policies from the organization level. When an administrator creates a new named policy, there are no defaults as in the Service Management pages. (There are no default values supported for named policy objects.)

Policy Service XML

In DSAME 5.0, the policy schema and default values should be defined by service developers in a service XML file. A sample policy schema definition is provided in the `sampleMailService.xml` provided in the DSAME product in the `<dsame_root>/SUNWam/samples/admin/cli/sampleMailService`. (See Section “Description of sampleMailService Files” on page 160 for some information and an example of how policy schema would be defined in a service XML file.

In DSAME 5.0, the policy.xml and policy.dtd are not public, thus should not be used to define policy schema and configuration data for a service.

Roles and Policy (Aggregation)

There is also an option for aggregation. For example, you could have three roles: Eng_role, Manager_role, and Pubs_role. They could all have different policies. When users possess multiple roles, the roles’ privileges must be aggregated and returned to the user who possesses all those roles, so that the user gets an aggregation of that role’s privileges.

If an administrator goes to a role, and enables a service for the role, he or she could pick any one of the services, configure different values for the service’s attributes, submit the new configuration, and then gets a CoS template. An administrator could create multiple services, and then assign a particular user a role or multiple roles. When DSAME gets this entry, the way CoS and roles works is that the

Directory server finds the role, sees that it has a CoS entry, and adds that to the attributes that get returned to the user. If there are any priority conflicts, it will look at the priorities for each one, and gets the priorities based on the CoS attributes for the group.

The service template is a set of attributes. For instance, if the highest priority was a set that was assigned to your role, that group of attributes is what you would get.

Overview of User Management Module

The User Management module is one of three modules, or components, within *Services Management i*n DSAME 5.0. It is contained, along with the Policy Management module, the Service Management Module, under a grouping of services called *Services Management.*

The User Management component of DSAME provides interfaces for creating and managing user-related objects in Directory server. All of the user objects that DSAME can read, get and store are:

- Organization
- Sub-Organization
- Organization Unit
- People Container
- Group
- Role
- User

The generic management functions that can be performed are create, delete, get, add, modify, and remove attributes on these objects, in addition to object-specific operations.

For information on when and how you can customize the ums.xml configuration file, refer to the section Section "Cases where Service Developers must Modify the ums.xml Configuration File" on page 130, and to the section on installing against a legacy DIT in the *iPlanet Directory Server Access Management Edition Installation and Configuration Guide.*

Adding User Attributes to DSAME

Any attribute that can be in the User entry can be managed through the User page in DSAME. To do this, you need to add the new user attribute to the `amUser.xml` file. Following is an example how to add the `nsaccountlock` attribute to the User profile.

NOTE You can take this same approach to set many attributes for the user. In this case, you could also group the attributes into a service. If you grouped attributes into a service, you would need to create a new XML service file. However, the simplest way to add an attribute is just to extend the `amUser.xml` file.

Following are steps to do this:

1. Add the following to `/opt/SUNWam/web-apps/services/WEB-INF/config/xml/amUser.xml`

   under the <SubSchema name=”User”> element, add the following attribute schema definition:

Code Example 5-5 Attribute Schema Definition to Add to amUser.xml File

```
<AttributeSchema name=nsaccountlock
    type="single_choice"
    syntax="string"
    any="filter|display"
    <ChoiceValues>
    <Value>true</Value>
    <Value>false</Value>
    </ChoiceValues>
</AttributeSchema>
```

2. In the User schema definition section, add the following:

```
<AttributeValuePair>
    <Attribute name="nsaccountlock"/>
    <Value>false</Value>
</AttributeValuePair>
```

   This will be the default value.

3. Also update the following file

`/opt/SUNWam/web-apps/services/WEB-INF/classes/amUser.properties`

with the new i18n tag “u13”.

```
u13=User Account Locked
```

or whatever text you want displayed on the User Properties page in DSAME console.

4. Then go to the Directory Server Console and remove the following OrganizationalUnit entry:

`ou=iPlanetAMUserService,ou=services,dc=sun,dc=com`

and its subentries.

NOTE In the DSAME 5.0 release, you can use the `amadmin` CLI tool to remove a service schema; you can also remove a service schema by using Directory Console.

5. Delete the user service using the `amadmin` tool.

```
amadmin --runasdn
"uid=amadmin,ou=people,o=iplanet.com,dc=sun,dc=com" --password
11111111 --deleteservice amUser.xml
```

Then reload the user schema using the `--schema` option of `amadmin`:

```
amadmin --runasdn
"uid=amadmin,ou=people,o=iplanet.com,dc=sun,dc=com" --password
11111111 --schema amUser.xml
```

6. Restart the `amserver`.

Now when you go to the User profile in DSAME console you will be able to set this attribute and lock the user out.

Defining User Attributes in a Service

User attributes are attributes that will only be part of the user entry. User attributes are not inherited from the role, organization, or sub-organization levels. A user attribute is any attribute that is typically different for every user or for which it does not make sense to put into a role or organization. If you define an attribute as type "User", it cannot be set in a role or organization and then inherited by the users who possess that role or who are in that organization.

Examples of user type attributes are:

- userid
- employee number
- passwords

There are no global defaults for User attributes.

User is considered a Service in DSAME

DSAME has several services, one of which is a User service. The User service is a group of attributes that are defined for and pertain only to a specific user.

A user attribute is any attribute that will show up on the User profile page.

There are no global defaults for User attribute types.

Customizing User Pages

To customize user pages in DSAME console, you would need to modify the `amUser.xml` file. This file is where user attributes are described. This file describes the User Service for DSAME.

NOTE	Any service can describe an attribute that is for a user only. This file is just the default placeholder for user attributes that are not tied to a particular service.

When displaying a user's attributes, the DSAME console gets all attributes from all services that are of SubSchema type User and displays them. For example, a few attributes from the `sun-user` object class are added so that a new service need not be created. You can just modify (or extend) the `amUser` service.

Extending what DSAME displays on the User Page in DSAME console

The User page and what it displays will vary, depending on what the administrator wants the users to see. By default, you can see a list of several attributes, which are retrieved from the `amUser.xml` file. Every attribute in the `amUser.xml` file that has an i18N display tag and the `any` attribute is set to display (that is, `any=display`) would display in DSAME console.

For example, you could have a hundred attributes in your profile, but if you only have seven attributes defined in a service XML file that have the `any` attribute set to display (`any=display`), you'll see just those seven attributes displayed in DSAME console. What that means is that if somebody adds their own auxiliary classes to the User service XML (`amUser.xml`), and they want them to display in DSAME console, they would have to update that XML file (`amUser.xml`). This is one way to extend what DSAME displays in the DSAME console.

The second option is to show the User attributes. In every service XML that is defined, if the attribute is specified to be User, such as in the mail service, then what the DSAME console does is display the User profile. If two of the attributes are not specified to display on the Create pages (with the `any=display` setting in the service XML file), it will display the attributes that are in the User profile, plus it will display the ones that are in all the other user service attributes, or tags. This provides the administrator with several options.

The DSAME console has a list of all the services and all the meta-data. When it gets all the attributes from the user profile (because it's CoS-related, it's getting them all in one call); it matches them and then it puts them into service categories. These are the attributes that are in the User XML (`amUser.xml`), which are attributes defined just for the user.

You could group all your attributes and then add them to the user and say, "these are the ones I want to display in DSAME console."

How the "any" Attribute can be used in Service XML Files

The `any` attribute and how it can used in service XML files is described in this section.

As a service developer, you not want to have administrators and users have to fill in all the fields to create a user. To specify which attributes will display in the Create, Properties, and Search pages in DSAME console, the "any" attribute can be specified for each attribute in a service XML file. For example, currently, in LDAP,

you have object classes. There are mandatory attributes with which to create objects in Directory server. You can specify when you create a class that certain attributes are required, that is, the administrator must enter a value for that attribute in DSAME console.

Specifying the "any" attribute in a service XML file is a way for the service developer to specify what he or she wants to display on the Create page (and Properties and Search pages) in DSAME console when a user is created. If you have 75 attributes in your entry, you might not want to type in all 75 attribute values. You may just want to enter username and password. So by adding this the "display" keyword to your attribute (`any=display`), when the Create window displays, this attribute will display. So—uid, password, fullname, last name—those have required attribute types. As long as they're defined in the schema, you can just put "required" and they'll display on the Create page.

The "any" attribute can be set to:

- required—attribute displays on Create page, and is a required attribute
- optional—attribute displays on Create page, but it is not a required attribute
- filter—attribute will display in the Filter pages
- display—attribute will display in the Create page

If the "any" attribute is set to a value of "required," then it shows up on the Create (user) page. For example, when you create a user, the Create page shows up with a number of fields to fill in. DSAME gets those fields (attributes) from the User XML (amUser.xml) file, and if the any attribute has a value or string, and "any = required," it displays on the Create page.

The any attribute is typically used in the any User type attribute, for example, in the amUser.xml or some other service XML file in a user schema definition. You typically would only see the any attribute in a User schema definition for any user service, for example, a mail service. See the Code Example 5-6 on page 123 for an example of the any attribute setting.

Code Example 5-6 Excerpt defining the any attribute in the amUser.xml File

```
<Dynamic>
     <AttributeSchema name="preferredlanguage"
              type="single"
              syntax="string"
              any="display"
              i18nKey="d1">
         <DefaultValues>
                  <Value>en</Value>
         </DefaultValues>
```

Code Example 5-6 Excerpt defining the any attribute in the amUser.xml File

```
        </AttributeSchema>
             <AttributeSchema name="preferredtimezone"
                     type="single"
                     syntax="string"
                     any="display"
                     i18nKey="d2">
        </AttributeSchema>
```

The any attribute has four possible values:

- any=required | optional | display | filter

The required and display can be specified together with a pipe symbol separating the two keywords (any=required | display), and the optional and display keywords can be specified together (any=optional | display).

The required and optional settings relate to the Create pages in DSAME console. If the any attribute is set to either required or optional, then when a user entry or organization, for example, is created, DSAME displays that attribute on the Create page. If the any attribute is set to required (any=required), an asterisk will display in that attribute's field, and a value must be entered for the object to be created.

If the any attribute is set to optional (any=optional), it will display on the Create page, but users are not required to enter a value; it's optional.

If the any attribute is set to display (any=display), then when the properties are display for the Create page, this attribute is displayed.

If the any attribute is set to filter (any=filter), when you go to the User page, and click Search, the attributes display for the filter. The any=filter setting lets service developers specify if this attribute should appear on this page. For example, if a service developer wanted to add employee_name, or manager_name to the default Filter page for the user, he/she would specify that attribute in the user service or any User schema definition in any service XML file as "any=filter". Then this attribute would display on the Filter page.

The any=display attribute type simply indicates whether DSAME should display that attribute or not (any=display), and whether the user must enter a value for the attribute (required), or if it is an optional attribute (optional), or whether that attribute should appear on the Filter page in DSAME console (filter).

The any attribute gives service developers some control over which attributes to display in the DSAME console. It is not necessary to display every attribute in all of the service XMLs on the DSAME console. Also, depending on the level of permissions you have, or whether the level of administrator you have access to may not allow you to see certain attributes on certain pages.

If you had an attribute that you didn't want users to see in the DSAME console, but it was still in their profile, you could the any attribute for that attribute to any=display. Or you could make it blank or empty (" "), and no attribute would display on the Create page.

In summary, the any attribute is useful if you want to extend the User page with additional attributes. Any time a service developer adds a new or custom service, he/she might be adding some new or modified objectclasses and attributes. In this case, he/she would probably want some of these new or modified attributes to appear on the User page. Use the any attribute in the service XML files to do this.

Extending the amEntrySpecific.xml File

These same rules apply for the `amEntrySpecific.xml` file for any of the DSAME abstract types, such as organization, organizational unit (ou), group, etc. This is the only other place you will see the any attributes used. The any attribute is used for creation and filter purposes, and it applies to any attribute in a User schema definition, either in the amUser.xml file or some other User schema definition in a service XML file.

Adding attributes Common to all Users to the User Service in DSAME

One way you can customize the User service in DSAME, for example, is to add any additional object classes with attributes to all user entries in the Directory server. This can be done through `ldapmodify` or through Directory Console.

For example, you might have an auxiliary object class with 10 attributes common to all user entries in their Directory server. To manage these attributes through DSAME they must be described in a service within DSAME. One option would be to add them to the iPlanetAMUserService (amUser.xml) service. This is the service within DSAME that, by default, includes many common attributes from the inetOrgPerson and inetUser object classes. This User service can be extended to include the attributes in the customer's object class or, alternatively, the customer might choose to create a new service to manage these attributes. Whether an existing service is extended or a new service is created, the attributes, after being imported would be manageable through the DSAME console.

It is important to understand that adding a service to DSAME enables DSAME to manage the attribute values from the DSAME console. It does not implement any behavior behind those attributes or dynamically generate any code to interpret the attributes. This means that DSAME can set and get the values; however, it is up to an external application to interpret or use these values. Continuing with the example above, if attributes common to all users were added, an administrator would be able to get or set those values or add them to roles or organizations. For example, one of the applications that would use DSAME might be a phonebook application that is reading those values from the Directory server. An administrator might create users, and set those new attributes through the DSAME console, but the phonebook application would be reading those attributes when querying for specific users.

Take note that in the DSAME console under Service Management there are several DSAME (internal) services shipped with the product. These services are managed in the same way as external services; the difference is that DSAME provides code implementations that use the service attributes. A good example of this is the DSAME URL Policy Agent service. This service defines three attributes that are used by the URL Policy Agent service (web agents) to check user access to URLs. The DSAME console allows the administrator to configure these attribute values, but the web agent is the external DSAME application using those attributes. The core DSAME features are each described and managed as any external service would be managed. You can view the service XML file that describes each DSAME service in the XML directory under `/SUNWam/web-apps/services/WEB-INF/config/xml`. This is a good place to start when describing or adding a service to DSAME.

Customizing Organization Pages

If customization engineers or service developers wanted to customize the organization pages, or pages for any of the other DSAME abstract types (such as organization units, people containers, static groups, filter groups, assigning dynamic groups, they would need to make modifications to the Entry-Specific service (specified by amEntrySpecific.xml).

The attributes in the `amEntrySpecific.xml` file are for console display purposes. This service (defined in the `amEntrySpecific.xml` file) is used by the DSAME console when customizing the display for specific DSAME abstract types, which are:

- organization
- container (organizational units)

- people container
- Static Group
- Filtered Group
- Assignable Dynamic Group

The DSAME types are manageable entities which are identified by the existence of DSAME marker object classes in directory entries. The marker object classes allow DSAME to manage most directory structures, regardless of the object classes and naming attributes deployed. For example, a directory may use only organizationUnits for their first level structure. By adding the DSAME Organization marker object class to these organizationalUnits, DSAME will manage them as DSAME organizations. Or user entries in the directory may contain only customer-specific object classes. By adding the DSAME User marker object classes, DSAME can manage these users. For more detailed information on the DSAME entry types and their marker object classes, refer to information on DSAME entry types and markers in the *iPlanet Directory Server Access Management Edition Installation and Configuration Guide.*

Each DSAME entry type will appear on the DSAME console in three places where this service is used:

- create page
- properties page
- search page

Each DSAME entry can have its own subSchema in the `amEntrySpecific.xml` file. If the DSAME type does not appear in the XML, the defaults will be used for the type. The default means that only the name of the entry will appear on the Properties display, the Create window, and the Filter window in the DSAME console. For example, the Organization type has the following XML subSchema.

Code Example 5-7 SubSchema for Organization entry type in 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>
```

Code Example 5-7 SubSchema for Organization entry type in amEntrySpecific.xml

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

Purpose of amEntrySpecific.xml File

When an object of one of the above (abstract) types is displayed in the DSAME console, the XML (amEntrySpecific.xml) describes which attributes will display on the Properties page, Create page, and Search page, if applicable.

If your DIT had customized the OrgUnit, Group, Role, or People Container user objects, you would have to add or modify their schemas also in the amEntrySpecific.xml file.

Note that the User is not configured here but in its own `amUser.xml`.

All the attributes listed in the schema are displayed when the organization pages (or pages for some other DSAME abstract entry such as organizational unit, static group, filtered group, managed group, people container, and role) are displayed. If the attribute is not listed in the schema definition in the `amEntrySpecific.xml` file, the DSAME console will not display the attribute.

any Attribute

The any attribute can have two of the four possible values:

- display, and
- required, or
- optional, and / or
- filter

For example, `required | filter` or `optional | filter` can be specified.

The `required` and `optional` values tell the DSAME console whether the attribute should be on the create page. If its keyword is `required`, it must have a value or the DSAME console will not allow the creation to occur. The filter tells the console whether to use this attribute in the search page or organizations. For example, if you wanted both attributes to be displayed on the Organization page, both are required for creation, and only the attribute will be used in the search page.

The `filter` keyword tells the DSAME console whether to use this attribute on the Search page or on the Organization pages. In the example above, it is desired to have both attributes display on the organization page. Both are required for creation, and only the attribute will be used on the Search page.

Type Attribute

The `type` attribute may be one of string, string list, single_choice, multiple_choice, or boolean. The `type` attribute specifies the format of values that this attribute can have. This can be one of the following types:

- string
- string list
- single choice
- multiple choice, or
- boolean

For example if the `sun-org-city` attribute can only be one of Concord, San Francisco, or Palo Alto, you would make this attribute a *single choice* and the cities above would be the choices. The DSAME console would display this as a list with only those cities.

If multiple cities were allowed, the attribute could be a *multiple choice*.

The purpose of this file is to display DSAME abstract entries, such as Organization pages (or organization unit, group, role, or people container). Typically, a service developer would be customizing the organization pages, however.

The DSAME console uses the data in this file for display purposes. Each DSAME abstract entry can have a schema definition in this XML file. For this example, you only need to modify the Organization schema. If a service developer or customization engineer had customized the OrgUnit, Group, Role or People Container in the DIT, he/she would have had to add or modify their schemas also.

Note that User is not configured here but in its own `amUser.xml`.

All the attributes listed in the subschema are displayed when the organization is displayed. If the attribute is not listed the console will not display the attribute.

Cases where Service Developers must Modify the ums.xml Configuration File

The ums.xml configuration file contains all the user (and service and policy) objects that are stored in Directory server. The ums.xml is called the DAI service in Directory server. Generally speaking, service developers or customization engineers will not need to know about or modify the ums.xml configuration file; only service developers or customization engineers with a good understanding of this file should make any modifications to it. Information on when and how to modify the ums.xml configuration file is covered in the section on installing against a legacy DIT in the *iPlanet Directory Server Access Management Edition Installation and Configuration Guide*.

The only cases where customization engineers or service developers will need to know about and modify this file are:

- when adding new objectclasses to users or organizations.
- if service developers do not want the default organizations or roles, and want to change them.
- if service developers want to change the naming attribute for an attribute.

What DSAME Supports in the Service Registration DTD

DSAME 5.0 only supports a limited subset of elements and attributes from the sms.dtd (Services Management Service Document Type Definition). The sms.dtd should not be modified, as the XML parser uses this file exactly as is; this file is meant to be used as a guide when writing service XML files. Additionally, the sms.dtd enforces certain rules when writing service XML files to be used by DSAME. This section lists the elements and attributes that DSAME 5.0 supports.

In DSAME, there are five support schema types:

- global
- organization
- dynamic
- policy
- user

In your service XMLs, service developers must describe the service attributes to be one of these types. For each schema type, you must define an attribute or attributes. The schema for each attribute can define the following attributes and keywords defined in this section:

- name
- syntax—The syntax attribute specifies the syntax of the attribute value. Currently supported syntaxes are:
  - boolean
  - string
  - password
  - number
  - any
  - i18NKey
  - DefaultValues element
  - ChoiceValue element

Each service also has three global attributes that are defined:

- i18NFileName
- i18nKey
- optionally may have service objectclasses;

Within each of the schema attribute types (global, org, dynamic, policy, and user), you will define your attribute schema, and one or more attributes. For each attribute, you need to define the type, syntax, i18Nkey, default values or choice values, if it's a type of single or multiple choice, and possibly the "any" attribute.

Globally for the service, you need to define the name of the service, and the version (which should be the same). The i18NFileName attribute is used to pick up the properties and all its indexes, or keys; then the key itself for the name of the service (i18Nkey for the name of the service). Optionally, a service developer could define one global attribute that describes the LDAP object class. (This LDAP objectclass would only need to be specified when the attribute is of the type dynamic or policy. If the attribute is global or organization, then specifying the LDAP objectclass is optional; this is because the global and organization attributes are private data, and are not stored in Directory server as LDAP attributes.)

Code Example 5-8 Excerpt from sampleMailService.xml showing global attribute definitions

```
<ServicesConfiguration>
    <Service name="sampleMailService" version="1.0">
        <Schema
            i18nFileName="sampleMailService"
            i18nKey="iplanet-am-sample-mail-service-description">
   <Global>
            <AttributeSchema name="serviceObjectClasses"
                type="list"
                syntax="string"
                i18nKey="">
                <DefaultValues>
                    <Value>iplanet-am-sample-mail-service</Value>
                </DefaultValues>
            </AttributeSchema>
```

The rest of the sms.dtd elements and attributes are reserved for future use by DSAME.

Service Schema Definitions Supported by DSAME 5.0

This section provides brief definitions of the schema attributes and elements in the sms.dtd that are supported by DSAME 5.0. Services that require their configuration information to be managed through the interfaces provided by the DSAME SDK and the DSAME console must organize this information using the DTD (sms.dtd) described in this chapter.

NOTE Only a small percentage of the elements and attributes in the `sms.dtd` are supported by DSAME 5.0. This chapter provides some information on the elements and attributes used by DSAME 5.0.

The following attributes and elements in the sms.dtd are supported and used by DSAME 5.0:

- *i18nFileName* which gives the name of the .properties file (for example, dpAuth.properties).
- *i18nKey* which gives the I18N index key that describes the service for a given locale.

The element *AttributeSchema* defines the schema of a configuration parameter. Using the *AttributeSchema* element, service developers can define the XML schema for configuration parameters by having multiple *AttributeSchema* elements. The service registration DTD (sms.dtd) provides several attributes to describe the schema for the attribute:

- *name*: specifies the name for the attribute (for example, mailQuota).
- *type*: specifies the format of values that this attribute can have. Currently-supported formats are:
  - *single* which specifies that the attribute can have only a single value;
  - *list* which specifies that the attribute can have multiple values;
  - *single_choice* which specifies that the attribute can have a single choice that must be selected from a choice list;
  - *multiple_choice* which specifies that the attribute can have multiple values from which one choice must be selected from the choice list. The default value for type is *list*.
- *syntax*: specifies the syntax of the attribute value. Currently-supported syntaxes are:
  - boolean
  - string
  - password
  - number
- *any*: has four possible settings—required, optional, display, and filter. Used to determine how to display a user entry or organization or some other type of entry on the Create, Properties, and Search pages in DSAME console.

In addition to the above attributes for the element *AttributeSchema*, it supports the following elements:

- *DefaultValues*: provides the default values for the configuration parameter.
- *ChoiceValues*: provides the possible choice values for the configuration parameter if its type is either *single_choice* or *multiple_choice*.

NOTE Custom pluggable authentication services that you can write and plug into DSAME will typically only use global and organization service attributes in their service XML files. Service XML files only need to be written and imported for custom pluggable authentication services if you have attribute/value pairs that you want to be manageable from DSAME console; otherwise, it is not necessary to write and import a service XML file. (See Chapter 2, "Pluggable Authentication SPI for detailed information on writing a custom pluggable authentication service and integrating it into DSAME.)

NOTE Default values specified in the service schema section of a service XML file will display for that attribute or configuration parameter in DSAME console, which the administrator can then change when creating a service template.

Attributes and Elements that DSAME Supports

This section lists the elements and attributes that DSAME 5.0 supports and uses. Note that in DSAME 5.0, only a subset of the elements and attributes in the sms.dtd are supported. The rest are reserved for future use. These are all the attributes and elements that service developers and customization engineers can use when defining service XMLs.

Purpose of an XML DTD

The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements. A DTD can be declared inline in your XML document, or as an external reference.

XML attributes are normally used to describe XML elements, or to provide additional information about elements.

The Services Management Services DTD is designed specifically for use with DSAME product, and other Sun service management products. It should not be altered; otherwise, DSAME service attributes may not be read and written correctly by iPlanet Directory server.

Where you can find Further Information on XML and DTDs

Here are a couple of sites that have information on XML and DTDs:

`http://www.xml101.com`

Additionally, there are several books and web site sources on XML (Extensible Markup Language) which explain standard XML terms, such as elements, attributes, and other XML keywords such as CDATA and NMTOKEN. These can be found by using any standard search engine.

Description of sms.dtd Elements and Attributes

The `sms.dtd` is a Document Type Definition that defines a set of rules that all service XML (Extensible Markup Language) file in DSAME must follow. It explains data structures that are used by all services in DSAME (whether internal to DSAME, that is, services that exist when you install DSAME; or external, that is, those services or applications that customers would integrate into DSAME).

A DTD (Document Type Definition) defines schema that will be used by services to define their configuration data. In other words, it's a sort of "schema for schema". The `sms.dtd` defines the schema that will be used by services to define their configuration data, and is specific to DSAME.

ServicesConfiguration Element

ServicesConfiguration is the root node for all services' configuration parameters and tasks in the entire XML document. You can describe and register multiple services in a single XML file. Within the ServicesConfiguration element you can define service schema for one or more services.

Code Example 5-9 ServicesConfiguration Element

```
<!ELEMENT ServicesConfiguration ( Service )+ >
<!ATTLIST ServicesConfiguration
    version    NMTOKEN     "1.0"
```

Schema Element

Before describing each attribute type, a schema element definition should be included in the service XML.

Code Example 5-10 Schema Element definition

```
<!-- Schema defines the schema for the configuration parameters
of the service. The sub-elements provide configuration
parameters' schema for the respective parameter grouping. The
attributes provide information for I18N, that is, properties file
name and the URL of the jar file which contains the properties
file. -->
<!ELEMENT Schema ( Global?, Organization?, Dynamic?, Policy?,
User? ) >
<!ATTLIST Schema i18nJarURL CDATA #IMPLIED
    i18nFileName CDATA #IMPLIED
%i18nIndex; >
```

Service Element

The *Service* element defines a service-specific task, configuration parameters' schema, and / or configuration data. Examples of services (or components) are Authentication, Session, Logging, User, URL Policy Agent, etc. (The word service and component are used interchangeably.) The attribute name gives the name of the product (can be a service, component, application, etc.) and version specifies the version of the product.

Service Name and Version Attribute List

For the Service element definition, you can define one or more schema, and 0 or more configurations. The attribute list gives the service name, and version.

Note that when writing a custom service, you should define only one service per XML file.

Code Example 5-11 Service Element Definition

```
<!ELEMENT Service ( Schema?, Configuration? ) >
<!--Name and version of the service -->
<!ATTLIST Service name NMTOKEN #REQUIRED
    version NMTOKEN #REQUIRED >
```

The attribute list (ATTLIST) defines the *Service name* and *version*. (See the *Service name* and *version* definitions in the excerpt from the `SampleMailService.xml` file in Code Example 5-14 on page 139.) It defines the service's name and version of that service.

Code Example 5-12 Service Element Definition from sampleMailService.xml

```
<ServicesConfiguration>
    <Service name="sampleMailService" version="1.0">
        <Schema
```

The version number does not need to correspond to the version number of the product. For example, the product version could be DSAME 5.0, and the Service name and version could be "DSAME" and "2.0", respectively, for example. The service version specified in the `sms.dtd` and the product version do not necessarily correspond to each other.

Service Name Attribute

This is a required attribute; you must define this in a service XML.

The "Service name" attribute for the Service element defines the name of the service being defined. For example, in the sampleMailService.xml file, the service name is "sampleMailService".

i18nFileName Attribute

This is a required attribute; you must define this in a service XML.

The i18nFileName attribute defines the localization filename to be used; the i18nFileName is also called the localization .properties filename. In the sampleMailService.xml file, the i18nFileName attribute is defined as "sampleMailService". In the case of the sampleMailService, the i18nFileName is defined as sampleMailService.properties.

i18nKey Attribute

This is a required attribute; you must define this in a service XML.

The i18nKey attribute defines the index key, or the "localization key" which displays the service name. In the sampleMailService.xml file, the i18nKey attribute defines the service name that will display in the DSAME console—"iplanet-am-sample-mail-service-description".

When looking at a service XML file, you can see that the service name is iPlanet logging service. When looking at the service schema definition section for the logging service, you can see that each attribute description of an i18n description key. These `i18nkey` attributes are localization keys (for example, "a1", "a2", "a3", etc.), and correspond to actual fields in each service's properties files. These fields are picked up and displayed by the DSAME console or some administrative application, based on the locale for that platform. Every attribute must be displayed in a particular locale for that platform language, for example, American English, Japanese, German, etc. In the resource bundle .properties file for every service (for example, amAuthLDAP.properties and amLogging.properties files). You can look at the services' .properties file to get the name of the actual attribute as it will display in DSAME console. This key corresponds to the actual attribute that needs to be changed on the server.

i18NKey Attribute and i18NFileName Attribute

The i18N key is a localization key, or index key, that maps to the localization file to be used. i18N is the localization file, for example, i18NFileName for the user, for example, `i18nFileName="amUser"`. The DSAME console retrieves an i18N key, for example, "u1", and opens the file `amUser.properties`, for example. See Code Example 5-13 on page 138 for a sample i18NKey definition. Also, when adding new attributes, you need to define an i18N key. If there is no description or a blank (that is, " "), that tells the DSAME console not to display the attribute.

Code Example 5-13 Excerpt showing the i18NKey Attribute Definitions in sampleMailService.xml

```
<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>
```

Note that the values you specify for the i18Nkey attributes in service XMLs determine the order in which the fields are displayed on a service page in DSAME console. To cause the attributes to display in alphabetical order, give the i18NKey attribute values "a1", "a2", and "b1" and "b2", and so forth. For example, if you add a new attribute and you want it to display at the top of the services page, make sure you give the i18NKey a value of "a1".

The i18n attribute points to a Java.properties file. Calls are made to the resource bundle and Java classes. It is defined in the class path, and then DSAME loads it. Depending on what locale is passed to it, DSAME will look in the right directory. Depending on what locale the customization engineer has set, the correct resource bundle is passed to it.

See Code Example 5-14 on page 139 for an example of the i18N key and i18FileName attribute definitions in the sampleMailService.xml file.

Code Example 5-14 i18NKey and i18FileName Attribute Definitions in sampleMailService.xml File

```
<ServicesConfiguration>
    <Service name="sampleMailService" version="1.0">
        <Schema
            i18nFileName="sampleMailService"
            i18nKey="iplanet-am-sample-mail-service-description">
   <Global>
            <AttributeSchema name="serviceObjectClasses"
                type="list"
                syntax="string"
                i18nKey="">
                <DefaultValues>
                    <Value>iplanet-am-sample-mail-service</Value>
                </DefaultValues>
            </AttributeSchema>
   </Global>
```

Global Attributes

A service developer typically must define the following global attribute types in a service XML file:

- name of service
- version of service
- i18NFileName
- i18NKey

- service objectclasses (optional)

These are tagged as global attributes in the service XML, but it is important to note that these attribute definitions apply to and describe the service itself.

In the `sampleMailService.xml` file, there is a global attribute schema defined as "serviceObjectClasses". In on page 140, the service object classes specified under that attribute schema's values are given to any new user that gets created in DSAME (and Directory server). These are the object classes for the sampleMailService, which are given to any new user that is created, and the value of the global attribute is `iplanet.am.sample.mail.service` which will be added to all every user entry that inherits the service attributes for the sampleMailService.

For dynamic and policy and user attributes, the serviceObjectClasses attribute schema definition, if used, must correspond to Directory server attributes. Unlike global and organization attributes, you must have an objectclass definition for dynamic, policy, and user attributes. (Defining the serviceObjectClasses schema attribute for global and organization is optional.) This means that you must define the name of that objectclass under this attribute serviceObjectClass. The reason behind this is that when your service is registered for an organization, when you go to create a user underneath that node, those objectclasses are automatically put into that user object for you. The three attribute types dynamic, policy, and user map one to one to their attributes in Directory server.

If adding a new service, a customization engineer must update the Directory server schema with any new objectclasses and attributes that the new service has, if they are not already in Directory server. (This is typically done through Directory Console, or by loading in the schema definition in the form of an .ldif file that contains the new object classes and attributes.)

Also, note that you can define multiple values for the attribute schema name *serviceObjectClasses* in a service XML file; for example, you could define multiple objectclass values in addition to `iplanet-am-sample-mail-service`.

For example, after a new service has been added to DSAME and Directory server, any new users that are created in DSAME will automatically get those new objectclasses and attributes; however, you must modify the already-existing users in the DIT so that they can use the new objectclasses and attributes. It does not matter whether a user belongs to that service or not; the objectclass is added. The user can be created through DSAME console or through `amadmin`.

Code Example 5-15 A Global Attribute Schema Definition in sampleMailService.xml

```
    <Global>
            <AttributeSchema name="serviceObjectClasses"
                type="list"
```

Code Example 5-15 A Global Attribute Schema Definition in sampleMailService.xml

```
                syntax="string"
                i18nKey="">
                <DefaultValues>
                    <Value>iplanet-am-sample-mail-service</Value>
                </DefaultValues>
            </AttributeSchema>
    </Global>
```

There can be more than one global attribute schema definition in a service XML file. A global attribute schema definition could define port number, cache size, number of threads, or any attribute that applies across an entire DSAME configuration or instance, thus to all services and users for that DSAME installation.

Note that global (and organization) attributes are internal to DSAME, and should not be used in any custom or external services in DSAME 5.0, except in custom pluggable authentication services. Global and organization attribute data are stored in Directory server as blobs of data, but cannot be retrieved through LDAP commands in DSAME 5.0. (Global and organization attributes can be used in DSAME 5.0 in custom pluggable authentication modules, however,

Organization Attributes

Different organizations in a web hosting environment, for example, "coke" and "pepsi" would have different configuration data defined for each organization. In this scenario, a service developer would need to define different configuration data for each of the organizations. A typical organization attribute could define the location for log files.

An example of a custom service where you would use organization attributes would be a custom pluggable authentication service. In DSAME 5.0, the only custom service that can use organization and global attributes is a custom pluggable authentication service.

For example, a service developer might specify an authentication module such as an LDAP authentication service or a RADIUS authentication service as an organization attribute type in a service XML. Something like an authentication service would typically apply at an organization level.

Organization attributes can only be used when writing custom pluggable authentication services; they cannot be used by external services in DSAME 5.0. Global and organization attribute data are privately stored in Directory server as blobs of data, and are not retrieveable using LDAP commands.

The only attribute types that are stored as LDAP objects in Directory server are *dynamic* and *policy* and *user*. The global and organization attribute types could also be stored in a flat file implementation, as they are not stored in Directory server as LDAP entries.)

Dynamic Attributes

The dynamic attribute type can be used for any attribute that should be assigned to roles or organizations, or inherited by users, that is, any attribute that is applicable to all user objects, with respect to the service. These attributes are usually implemented as Class of Service, which is functionality provided by iDS 5.0. Examples of dynamic attributes would be status attributes, mail address, etc.

Typical dynamic attributes would be session attributes; for example, a dynamic attribute defined in a mail service would get picked up by all users that use that mail service. And, based on the role that a particular user may possess, they would pick up these new attribute/values.

Dynamic attributes use the Directory Server CoS and roles feature. (See the Directory server 5.0 documentation on advanced entry management for details on how Class of Service works.)

Policy Elements

Policy element types are intended to provide a group of actions (or privileges) that are specific to the service. Examples of actions are canForwardEmailAddress, canChangeSalaryInformation, etc. The schema of the configuration parameters is provided by AttributeSchema. The element ResourceName specifies if the service has resources associated with it, for example, URLs in the case of URL Policy Agent service.

Code Example 5-16 Excerpt from amPolicy.xml defining a policy element

```
<ServicesConfiguration>
    <Service name="iPlanetPolicyService" version="5.0">
   <Schema>
       <Organization>
      <SubSchema name="Policies">
          <SubSchema name="NamedPolicy"
         inheritance="multiple">
         <AttributeSchema
             name="xmlpolicy"
             type="single"
             syntax="string" />
          </SubSchema>
      </SubSchema>
```

Code Example 5-16 Excerpt from amPolicy.xml defining a policy element

```
<SubSchema name="Resources">
    <SubSchema name="ServiceType"
   inheritance="multiple">
   <AttributeSchema
       name="xmlresources"
       type="single"
       syntax="string" />
    </SubSchema>
</SubSchema>
```

You can create a policy, delete a policy, and assign a policy to roles by using the amadmin CLI tool. Policy schema definitions in sms.dtd use Class of Service (CoS).

Dynamic attributes are typically used for roles. All users belonging to a particular role will inherit all the attributes associated with that role. Every time DSAME goes to get a user object from the Directory server, if that user is part of a particular role defined as a "dynamic" attribute, that user inherits all the attributes associated with that role that he/she belongs to. By default, the user gets all these objectclasses and attributes.

User attributes

A *user* attribute type is for attributes that will be physically present in the user entry. User attributes are not inherited by roles or organizations. Examples of User attributes are password and employee ID.

Global Element, AttributeSchema and SubSchema Sub-elements

You can specify a Global element, which has two sub-element definitions—AttributeSchema and SubSchema, for which you can specify 0 or more definitions.

Code Example 5-17 Global Element Definition in sms.dtd

```
<!-- Global element provides grouping of configuration
parameters that are globally applicable to all instances of
its service. In the case of services that are grouped, these
configuration parameters are global to that group. The schema
of the configuration parameters is provided by
AttributeSchema, and if there is any necessity to sub-group
additional configuration parameters, they can be grouped using
the SubSchema element. -->
<!ELEMENT Global ( AttributeSchema*, SubSchema* ) >
```

Attribute Schema Sub-Element

AttributeSchema sub-elements enable a service developer to define a specific attribute schema, or configuration parameter. See Code Example 5-18 on page 144.

Code Example 5-18 AttributeSchema Element Definition (with Sub-Elements defined)

```
<!-- AttributeSchema defines a single configuration parameter for
a service. The attribute name gives the name for the configurable
parameter, type specifies whether the parameter is single-valued,
multi-valued, single-valued choice or multi-valued choice type;
syntax defines whether the parameter is boolean, string, numeric
or dn; rangeStart and rangeEnd provide the starting and ending
values for attribute syntax decimal_range and number range,
respectively; and any provides means for service developers to
add service-specific information. The elements IsOptional,
IsServiceIdentifier, IsStatusAttribute represent whether the
attribute is optional, a service identifier (CoS specifier) or
status attribute, respectievely. The elements DefaultValues
provides the default values for the parameter and ChoiceValues
provides the possible values for the parameter if it is of choice
type. The element Condition, if present, specifies boolean
operations, which determine if the attribute is valid based on
the current configuration data. If multiple Condition elements
are present, it is sufficient if at least one of them satisfy the
requirement (this provides OR implementation). -->
<!ELEMENT AttributeSchema (IsOptional?, IsServiceIdentifier?,
IsStatusAttribute?,
   ChoiceValues?, BooleanValues?, DefaultValues?) >
<!ATTLIST AttributeSchema name NMTOKEN #REQUIRED
```

Code Example 5-18 AttributeSchema Element Definition (with Sub-Elements defined)

```
    type ( single | list | single_choice | multiple_choice )
"list"
    syntax ( boolean | string | password "string"

    CoSQualifier( default | override |
            | merge-schemes )"default"
    rangeStart CDATA #IMPLIED
    rangeEnd CDATA #IMPLIED
    any CDATA #IMPLIED
    %i18nIndex; >
```

For the syntax attribute, the password type will encrypt and decrypt when stored or retrieved in Directory server. Because encryption is private in DSAME 5.0, the syntax attribute specified as password should only be used for DSAME internal services, and should not be used by external services. In DSAME 5.0, this syntax attribute keyword should only be used for custom pluggable authentication services.

Only the syntax types of boolean, string, and password are enforced in DSAME 5.0; all other syntax types (such as dn, email, url, numeric, percent, number, decimal_number, number_range, and decimal_range) are treated as strings. For example, if you put in a number as a string, DSAME will accept it.

Note that the CoSQualifier operationalal keyword is not supported in DSAME 5.0.

Service Sub-Schema Element

The service sub-schema element is only used in the amEntrySpecific.xml file in DSAME 5.0. This sub-schema element is private in DSAME 5.0, thus should not be used in any external service XML files in DSAME 5.0.

Service sub-schema can specify a subschema, for example, global information. It could define multiple sub-schemas, such as for a particular application that is defined. For example, logging for a Calendar application could be a separate sub-schema. Each application could therefore define its own way of logging. A netmail application could define a sub-schema, where there could be multiple instances of subschema, and define what attributes that sub-schema should use. For example, for different logging levels, a customization engineer could define choice values for different logging levels. For logging type, you could also define some choice values, for example, 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.

AttributeSchema Element, ChoiceValues, BooleanValues, and DefaultValues Sub-elements

The fourth, fifth and sixth sub-elements are ChoiceValues, BooleanValues, and DefaultValues. These are elements for which specific attributes can have either a multiple choice list of values from which the user can choose several values, a Yes / No (boolean) list to choose from, and some default values from which to choose. The attributes and values specified generate actual values that will appear on the DSAME console. See the amAuth.xml file for examples of these element types defined in a DSAME service XML.

AttributeSchema Attribute, name Attribute

The attribute schema defines the attribute schema *name*, which is NMTOKEN. The AttributeSchema *name* attribute is required. This attribute schema name specifies the name of the attribute, for example, *port_number* or *logfile_name*, for the configuration data.

Code Example 5-19 Excerpt from sampleMailService.xml File showing AttributeSchema *name* specification

```
<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 Element, Type Attribute

The attribute type can be a single attribute, a list of attributes, a single_choice attribute, or a multiple_choice list. In Code Example 5-19 on page 146, note that type is specified as `single_choice`, and the choice values are specified as active, inactivate, and deleted. Given this service schema and configuration data specification in the service XML file, the administrator or user must then choose one of those values for the "iplanet-am-sample-mail-service-status" on the DSAME console.

If the attribute is of the type `multiple_choice`, then the administrator or user can pick multiple choices (more than one) on the DSAME console for that attribute.

For the type attribute, you can specify different keywords: `single`, which specifies that the user can only specify one value; specifying `list` lets the user choose from a list of values; `single_choice` allows the user to pick one choice from a list of values; `multiple_choice` allows the user to pick several choices from a multiple choice list.

AttributeSchema Element, Syntax Attribute

For the `syntax` attribute, if boolean is specified, users can pick between (true / false) for that attribute in the DSAME console.

The syntax attribute lets you specify from among a list of 12 keywords:

Syntax Attribute, boolean value

Specifies that user or administrator must select True / False for the attribute.

Code Example 5-20 Excerpt from sampleMailService.xml showing *boolean* syntax specification

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

Syntax Attribute, string value

Specifies that user can specify any string value for that attribute.

Code Example 5-21 Excerpt from sampleMailService.xml showing syntax attribute with string value

```
 <AttributeSchema
name="iplanet-am-sample-mail-sentmessages-folder"
                 type="single"
                 syntax="string"
                 i18nKey="a3">
                 <DefaultValues>
                     <Value>MailSent</Value>
                 </DefaultValues>
             </AttributeSchema>
```

AttributeSchema syntax Attribute, password value

Specifies that user must enter a password, which will be encrypted.

Code Example 5-22 amAuthLDAP.xml showing syntax attribute with value of password

```
 <AttributeSchema name="iplanet-am-auth-ldap-bind-passwd"
                   type="single"
                   syntax="password"
                   i18nKey="a4">
          <DefaultValues>


<Value>AQAAKh7ai3zuzP8VryBzcPnXdRA9ukTY2gX6</Value>
       </DefaultValues>
                </AttributeSchema>
```

AttributeSchema Element, ChoiceValues Sub-element

If the ChoiceValues sub-element is defined in the AttributeSchema, then it is valid, provided the type attribute is specified as either `single_choice` or `multiple_choice`. The administrator or user must choose either a single choice or from a multiple choice list for that attribute in the DSAME console.

Code Example 5-23 Excerpt from sampleMailService.xml showing `type` attribute with `single_choice` value

```
<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>
```

AttributeSchema Element, syntax Attribute, boolean value

If the syntax attribute is specified as `boolean`, then the administrator or user must select either True / False or Yes / No for that attribute in the DSAME console. In the Code Example 5-24 on page 149, note that the syntax attribute is specified with a "boolean" value.

Code Example 5-24 Attribute Schema Element Specification with boolean syntax specified

```
    <AttributeSchema name="iplanet-am-auth-ldap-ssl-enabled"
                        type="single"
                        syntax="boolean"
                        i18nKey="a8">
            <DefaultValues>
                         <Value>false</Value>
            </DefaultValues>
```

Note that the default values specified in the AttributeSchema element definitions in the service XML files are the values that will display in the DSAME console for each attribute specified. In the Code Example 5-24 on page 149, for example, on the LDAP Authentication service page, the "Enable SSL to LDAP Server" field (as specified in the corresponding localization .properties file (`amAuthLDAP.properties`) will give a default value of *false*. The administrator can change this value when creating a new service template with different values, if desired.

AttributeSchema Element, CoSQualifier Attribute

There are different values that a service developer can specify for CoSQualifier attributes:

- default
- override
- merge-schemes

If the `default` value is specified, then the default value for the CoSQualifier attribute indicates that if there are two conflicting CoSQualifier attributes assigned to the same user object, the one with the lowest priority takes precedence. The lowest number, or 0, has the highest priority, or takes precedence.

If the `override` value is specified for the CoSQualifier attribute, then CoS overrides the user value. (The default behavior is for the user entry value to override the value for CoS.)

The `operational` value is not supported in DSAME 5.0.

If the `merge-schemes` value is specified, if there are two CoS templates assigned to the same user, then the CoS templates get merged so that the values get added together; then the user gets an aggregation of the CoS templates, or a union of them.

For example, if there is an attribute "A" with a value of "10" in one template, and another template "B" has a value of 20, the user would get both values 10 and 20 for the attribute "A".

AttributeSchema Element, any Attribute

Specifies a wildcard that can be used by services to declare additional configuration information. All the attributes listed in the subschema are displayed when the organization is displayed. If the attribute is not listed, the DSAME console will not display the attribute. For more information on the "any" attribute, see Section “How the "any" Attribute can be used in Service XML Files” on page 122.

The any attribute can have four possible values:

- display, and
- required, or
- optional, and / or
- filter

The `required` and `optional` keywords tell the console whether the attribute should be on the Create page. If its keyword is `required`, it must have a value or the DSAME console will not allow the create to occur. The `filter` keyword tells the DSAME console whether to use this attribute on the Search page or on the organization pages.

Organization Element

Organization element is used to specify configuration parameters for things like an organization's authentication. You can specify AttributeSchema and SubSchema elements.

Code Example 5-25 Organization Element in sms.dtd

```
<!-- Organization element provides a grouping of configuration parameters that
can be configured differently for various organizations. Examples are
parameters like organization's authentication mechanisms, logging information,
etc. The schema of the configuration parameters is provided by AttributeSchema
and if there is any necessity to sub-group additional configuration parameters
they can be grouped using the SubSchema element. -->
<!ELEMENT Organization ( AttributeSchema*, SubSchema* ) >
```

Dynamic Element

Lets service developer provide a grouping of configuration parameters that all users would inherit. Dynamic elements use CoS (Class of Service) as implemented by iDS 5.1. Examples of dynamic configuration parameters, or attributes, are status attributes or mail address, etc.

Code Example 5-26 amSession.xml File showing some attributes specified as Dynamic

```
    <Dynamic>
                  <AttributeSchema name="iplanet-am-session-max-session-time"
                      type="single"
                      syntax="number"
                      i18nKey="a1">
              <DefaultValues>
             <Value>120</Value>
              </DefaultValues>
                  </AttributeSchema>
                  <AttributeSchema name="iplanet-am-session-max-idle-time"
                      type="single"
                      syntax="number"
                      i18nKey="a2">
              <DefaultValues>
```

Code Example 5-26 amSession.xml File showing some attributes specified as Dynamic *(Continued)*

```
                        <Value>30</Value>
                </DefaultValues>
```

Policy Element

The Policy element lets service developer specify policy privileges or action names (with ActionSchema element). Examples of actions (or privileges) are canForwardEmailAddress, or can Change SalaryInformation. Privileges are get, post, and put. The element Resource Name specifies if the service has resources associated with it, for example, URLs in the case of the URL Policy Agent service.

Code Example 5-27 Policy element

```
<!-- Policy element provides grouping of actions (or privileges) that are
specific to the service. Examples of actions are canForwardEmailAddress,
canChangeSalaryInformation, etc. The schema of the configuration parameters is
provided by AttributeSchema. The element HasResourceNames specifies if the
service has resources associated with it, for example, URLs in the case of Web
Agent Service. -->
<!ELEMENT Policy ( ActionSchema* ) >
```

User Element

The User element lets service developer specify a group of configuration parameters that are applicable to user objects, only, with respect to the service. Examples are things specific only to a particular user, such as social security number, email address, etc. The User element uses the AttributeSchema sub-element, but does not use the SubSchema element.

Code Example 5-28 Excerpts from amUser.xml showing User Attributes specified

```
          <User>
                <AttributeSchema name="uid"
                    type="single"
                    syntax="string"
                    any="required|filter"
                    i18nKey="u1">
    </AttributeSchema>
                <AttributeSchema name="givenname"
                    type="single"
```

Code Example 5-28 Excerpts from amUser.xml showing User Attributes specified *(Continued)*

```
                    syntax="string"
                    any="optional|filter|display"
                    i18nKey="u2">
    </AttributeSchema>
.
.
.
        </DefaultValues>
                </AttributeSchema>
            </User>
```

Policy Management Module

The *Policy Management* module is one of three modules (along with *User Management and Service Management* module) that comprise the Management Services.

Overview of Some Policy Concepts and Terms in DSAME

Policy management deals with creating policies for services and how they are applied to a role or an organization.

Policy Schema

Policy schema is a set of all valid privileges and possibly optional default values for each of the privileges in the set. Policy schema can also contain information about the service itself. Each service that enforces policy must have a policy schema defined for it.

Named Policy and Assigned Policy

Policy constitutes a set of one or more pairs of {privileges, privilege settings}. For example, {*mailboxQuota, 100MB}* is a policy. {*allowURLList, "*.red.iplanet.com, *.eng.sun.com"*} is another policy. When a user or administrator creates a policy on the Policy page, that is called a "named policy".

Policy is the result of changing or setting the default values for privileges in the Policy schema. Administrators or service developers of DSAME use the policy schema to create policy for the relevant service by modifying any default values for the privileges in the policy schema and applying the resulting policy to a role, a user, an organization or an organizationalUnit. When a user or administrator takes that named policy, and assigns it to an organization or a role, then it can be called an “assigned policy”.

Policies are unlike service schema; there are no named service objects. However, there are named policy objects.

From the same policy schema, the administrator can generate different policies to apply at different levels, such as role, user, organization, etc. Policy schemas can be viewed as *metadata* for the privileges of any given service. Using policy schemas, an administrator knows the valid set of privileges for any given service.

For example, you could compare the policy schema with the schemas in the XML, LDAP, and RDBMS worlds. Before you create an XML document, you define the XML schema. Before you create relational tables, you define the database schema. Before you create LDAP entries, you define the LDAP schema for DIT entries. Similarly, before you create policies, you need to create the Policy schema for the service.

For example, a Policy schema definition is shown in the `amWebAgent.xml` file in Code Example 5-29 on page 154.

Code Example 5-29 Excerpt showing policy schema definition in amWebAgent.xml File

```
            <Policy>
                <ActionSchema name="permission"
                    type="single_choice"
                    cosQualifier="merge-schemes"
                    i18nKey="p1">
                    <ActionValue i18nKey="a1">
                        <Value>iplanet-am-web-agent-access-allow-list</Value>
                        <ResourceName/>
                    </ActionValue>
                    <ActionValue i18nKey="a2">
                        <Value>iplanet-am-web-agent-access-deny-list</Value>
                        <ResourceName/>
                    </ActionValue>
                    <ActionValue i18nKey="a3">

<Value>iplanet-am-web-agent-access-not-enforced-list</Value>
                        <ResourceName/>
                    </ActionValue>
                </ActionSchema>
            </Policy>
        </Schema>
```

Code Example 5-29 Excerpt showing policy schema definition in amWebAgent.xml File

```
    </Service>
</ServicesConfiguration>
```

The policy schema elements and attributes are not public in DSAME 5.0. However, note that in the DSAME 5.0 release, no external services can define and use policy schema definitions. So, an example from the `amWebAgent.xml` file is included here to provide some overview information on how to define policy schema.

Note that there is a difference between a Policy schema and a policy. Policy schema *must* declare all the valid privileges (or attributes) for the service. Policy need not declare all the valid privileges.

Scoping of Policies

Because policies for any service can be associated at the role, organizational Unit (ou) and / or organization levels, there could be cases when there will be conflicting policies for the specified service for a given user. Therefore, priority levels are associated with policies. Policies with higher priority take precedence over templates with lower priority. If conflicting policies have the same priority, the scoping results will be undefined.

Adding a Custom Service

After installing DSAME, you can add new, or custom, services to it later. However, you must add the new objectclasses and attributes manually to Directory server, by using `ldapmodify` or adding the new objectclasses and attributes through the Directory Console.

Note that in DSAME 5.0, you are limited to writing custom pluggable authentication services (which must use only the global and organization attributes in their schema definitions), and external services (other than custom pluggable authentication services) which must use only the dynamic and user attributes. LDAP can be used to retrieve data from the dynamic and user attributes in the user entry. In the case of the custom pluggable authentication service, there are methods in the pluggable authentication SPI that allow the retrieving of organization-based data.

So if you define a custom application or service (other than a pluggable authentication service) such as a phonebook application which has dynamic and user attributes defined in its schema, a service developer could import this custom service into DSAME using the `amadmin` CLI tool, configure the service in DSAME console, assign the service to an organization or a role, and then the phonebook application would use LDAP directly to read user attributes.

Following are some things to keep in mind when adding a new service:

- If a customer wants to add a new service, he/she may need to modify a schema definition in the `ums.xml` file, and then may need to modify a couple of other files (`amUser.xml` and/or `amEntrySpecific.xml` file). If you want to add an additional service that was not there are installation time, then you need to just add the objectclasses and attributes to the `ums.xml` file to enable user creation. (Refer to the Section "Cases where Service Developers must Modify the ums.xml Configuration File" on page 130 for some information. Also see the *iPlanet Directory Server Access Management Edition Installation and Configuration Guide* for information on modifying the `ums.xml` configuration file when installing against a legacy DIT.
- The User Management Services component in DSAME is considered a service. Default attribute/value pairs are provided for all of the user management services attributes in the `ums.xml` and `umsCompliant.xml` files. These are loaded into the Directory server when DSAME is installed.
- Whenever a customer adds a new service or modifies an attribute, the `ums.xml` file must be modified and reloaded into Directory server.
- When a customer installs the the DSAME product, the `ums.xml` schema and configuration data gets saved in the Directory server. If DSAME were to be un-installed, then the user management services schema and configuration information previously stored in Directory server would not get un-installed. If you were to reinstall DSAME, the user management services components would already be in the Directory server.
- Whenever you reinstall the DSAME product, then the `ums.xml` file gets saved in the Directory server. The `ums.xml` file is saved in the Directory Server Console as the *DAI service*.
- Once the `ums.xml` configuration file (called DAI service in the Directory server) has been loaded at installation time, it can only be modified using Directory server Console. You cannot reimport using the amadmin CLI tool.
- When adding a service (such as sampleMailService.xml), you must update the DSAME console hierarchy service attribute to get it to display in the service tree. For information on updating the service hierarchy attribute, see Section "Add the sampleMailService to the Service Hierarchy" on page 206.)

If creating a custom service, a service developer or customization engineer would typically do the following:

High Level Flow for Creating and Registering Services

Following is a set of high level steps to create, register, and activate services:

NOTE You can customize the existing internal DSAME services by adding new service attributes, modifying existing attributes, or you could create new services to work with DSAME.

1. Write the custom service or application, for example, a Calendar application or an authentication service.

   (For specific information on creating custom pluggable authentication modules, see the Pluggable Authentication chapter. For specific information on creating a customized single sign-on solution to work with your existing services and applications, see the Single Sign-On chapter.)

2. First create the service XML file for the service, which must conform to the DTD (sms.dtd). See Section "Attributes and Elements that DSAME Supports" on page 134 more information on the DSAME service management services DTD (sms.dtd).

3. Update the `amEntrySpecific.xml` file, if necessary, for any abstract DSAME entries that you want to be displayed on the DSAME console's Create, Properties, or Search pages. (See Section "Attributes and Elements that DSAME Supports" on page 134 for information.)

4. Update the `amUser.xml` file, if necessary. (See Section "Attributes and Elements that DSAME Supports" on page 134 for information.)

5. Use Directory Console to add any new or modified schema (objectclasses or attributes) to Directory server so that DSAME can manage those new attributes. To do this, you could use either `ldapmodify` to update the Directory server schema, or modify the .ldif file directly.

6. Make a copy of one of the resource bundle .properties files, for example, `amAuthLDAP.properties`, and modify it for your custom application or service. It should be located in the `classes` subdirectory. (See the Pluggable Authentication SPI or Single Sign-On chapters, respectively, for information on creating and integrating custom authentication and single sign-on solutions.)

7. Copy the new or modified .properties file for your service into the following directory:

   dsame_root / web-apps / services / WEB-INF / classes

   (For more details on loading and registering services, see Section "Guidelines for Loading Services into DSAME" on page 198.)

8. Import the service XML file into DSAME by using the `amadmin` CLI tool. After loading or importing a service XML into DSAME, all the services are loaded into a separate branch in iDS (under the ou=services branch).

9. Update the DSAME service hierarchy attribute. (See Section "Add the sampleMailService to the Service Hierarchy" on page 206 for information on how to do this.)

10. After you have imported or loaded the service(s) into DSAME, each organization can then register the service through the DSAME console. You can do this by going to the organization, selecting the service, clicking Add, then select the service that you want to register, then click Submit.

    The service is now registered in DSAME. Administrators can then create different service templates for role, sub-organizations, etc. through the DSAME console.

11. Activate the service in DSAME console. (Alternatively, you can activate the service by importing the `activateRequests.xml` file by using the `amadmin` tool.)

For more detailed information on importing, registering, and activating service XML files into DSAME, see Chapter 6, "Using the Command Line Interface.

Some Things to Consider When Creating a New Service

For example, say you want to define a custom authentication service. You would need to define the service XML file for the custom authentication application or service per the `sms.dtd`. The things that the authentication service typically would need are logging, i18n information, and session information. Each one of these (logging, session, and authentication) is a service. They must define what each service will need, what all the attributes are, and what service configuration the application will need.

For example, logging contains a subset of these values, and could contain maximum log size file, location of log file, etc. The customer would need to define their schema based on schema format. One of the attributes they would need to define is where the log file will be stored, and similarly to define another service template for authentication. This information, when uploaded into DSAME, is passed to the SMS APIs, and the SMS APIs uploads the information into iDS.

The default values for each service attribute defined in the schema should be included within each schema definition. The default values, along with the schema definition, can also be referred to as the "configuration data".

At the customer site, an administrator could change the default values set in the XML files to new values, then create a new service template that uses those new values. For example, say you wanted to change your log file location to another location. An administrator or user could do this in the DSAME console, then save the new service template, and then assign it to an organization or role.

In DSAME 5.0, you cannot create Directory server schema dynamically through the amadmin CLI tool. You must always create or extend the Directory server (LDAP) schema by using the `ldapmodify` command for each service being created in DSAME (see the section on sampleMailService example), or by creating an .ldif file in the Configuration / Schema page in Directory Server Console.

After a service has been created, registered and activated for an organization, if a new user is added to that organization, it automatically inherits all the organization's CoS / template values for the global and policy attributes of the service (for example, through a role association).

The SDK adds the object class for the registered services to the user entry when it is created. The object class is defined in the service XML file. Note this only happens if the user is created through the SDK.

On the other hand, if a new service is instantiated, existing users for the organization do not inherit the attribute values for that service.

If services are added after the user is added, the user entries must be updated some other way (for example, through using the `ldapmodify` or by creating or modifying an `.ldif` file in Configuration / Schema page in Directory Server Console. The SDK will not update them if services are added after the user has been added.

The solution is to add the objectclass or attributes of the new service to each existing user through an `ldapmodify` command (as you would do when using the sampleMailService example), or by creating or modifying an `.ldif` file through the Config / Schema page in Directory Server Console. Alternatively, you could use the migration scripts provided in DSAME 5.0 to update the user objects in the DIT with the new objectclasses and attributes.) For information, see the section on migration scripts in the *iPlanet Directory Server Access Management Edition Installation and Configuration Guide*.)

You do not need to restart the LDAP server when extending the schema to add or modify users / services / organizations / roles, either directly through the .ldif file or Directory Server Console, or by using the `amadmin` CLI tool.

For new services being imported to DSAME, you need to restart the `amserver` to add any i18nKey resource bundles (if required) to its classpath.

Description of sampleMailService Files

This section provides a description of the following sample mail service files that are provided with the DSAME 5.0 product:

- `sampleMailService.ldif`
- `sampleMailService.xml`
- `sampleMailService.properties`

The files are located in the `<dsame_root>/samples/sampleMailService` directory in the DSAME product. These sample files are provided so that you can use them as guidelines when creating your own custom services and applications.

There are several sampleMailServices files provided—an `.ldif` file to show some sample objectclasses and attributes to add to Directory server; an `.xml` file to show how a sample service could be written; and a sample `.properties` file to show how localization keys point to the the actual fields that display on DSAME console would be created.

In on page 165, the `iplanet-am-sample-mail-service-status` attribute is specified as a `dynamic` attribute, with a type of "single_choice", a syntax of "string", a list of choice values (Active, Inactive, and Deleted), and a default value of Active. On the DSAME console, after this service is loaded using `amadmin`, users and administrators will see this list of choice values for the `iplanet-am-sample-mail-service-status` attribute, and the default value of Active. The i18nKey value of "a1" points to an actual field defined in the resource bundle .properties file for the service. (In this example, the file would be `sampleMailService.properties`). The value for the "a1" definition in the `sampleMailService.properties` file is the actual field that displays for the `iplanet-am-sample-mail-service-status` attribute. In this case, the "a1" index key, or localization key, is "Mail Status", which is the field that displays for the `iplanet-am-sample-mail-service-status` attribute on the sampleMailService service page in DSAME console.

Excerpts of the sampleMailService sample files are given in this section.

sampleMailServiceSchema.ldif File

Typically, the customization engineer or deployment engineer will need to modify the necessary `.ldif` file or update the objectclasses and attributes in iDS Console directly before loading the modified or created service XML.

Following is the sampleMailServiceSchema.ldif file, which is located in the `<dsame_root>/samples/cli/sampleMailService` directory. It is provided so that you can use it as a sample, to create the LDAP schema for a service. You would typically need to extend the schema first by using the .ldif file as input to the `ldapmodify` command.

NOTE If Directory server already knows about the objectclasses and attributes that are in your service, you do not need to update Directory server. If any of your service or application's objectclasses and attributes are not already up-to-date in Directory server, then you must extend the schema, either by modifying the .ldif file, or by updating the schema manually in Directory Console directory.

This is an example of an `.ldif` file, which a customization engineer could create, either manually, or by adding any new schema objects (objectclasses and attributes) by using Directory Console, then generating an `.ldif` file by using the `db2ldif` command. Another option is to load the `.ldif` file, with the new objectclasses and attributes (for a new service, or a modified service, for example) into DSAME and Directory server by using `ldapmodify` to modify the existing iDS schema.

Refer to the Directory server 5.0 documentation for specifics on using `ldapmodify` and how to add objectclasses and attributes to the Directory server.

Code Example 5-30 sampleMailServiceSchema.ldif File

```
dn: cn=schema
changetype:modify
add:attributeTypes
attributeTypes: ( iplanet-am-sample-mail-service-status-oid 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' )
attributeTypes: ( iplanet-am-sample-mail-root-folder-oid NAME
'iplanet-am-sample-mail-root-folder' DESC 'iPlanet SampleMailService
Attribute' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 X-ORIGIN 'iPlanet Directory
Pro' )
attributeTypes: ( iplanet-am-sample-mail-sentmessages-folder-oid NAME
'iplanet-am-sample-mail-sentmessages-folder' DESC 'iPlanet SampleMailService
Attribute' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 X-ORIGIN 'iPlanet Directory
Pro' )
attributeTypes: ( iplanet-am-sample-mail-indent-prefix-oid NAME
'iplanet-am-sample-mail-indent-prefix' DESC 'iPlanet SampleMailService
Attribute' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 X-ORIGIN 'iPlanet Directory
Pro' )
attributeTypes: ( iplanet-am-sample-mail-initial-headers-oid NAME
'iplanet-am-sample-mail-initial-headers' DESC 'iPlanet SampleMailService
Attribute' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 X-ORIGIN 'iPlanet Directory
Pro' )
attributeTypes: ( iplanet-am-sample-mail-inactivity-interval-oid NAME
'iplanet-am-sample-mail-inactivity-interval' DESC 'iPlanet SampleMailService
Attribute' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 X-ORIGIN 'iPlanet Directory
Pro' )
attributeTypes: ( iplanet-am-sample-mail-auto-load-oid NAME
'iplanet-am-sample-mail-auto-load' DESC 'iPlanet SampleMailService Attribute'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 X-ORIGIN 'iPlanet Directory Pro' )
attributeTypes: ( iplanet-am-sample-mail-headers-perpage-oid NAME
'iplanet-am-sample-mail-headers-perpage' DESC 'iPlanet SampleMailService
Attribute' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 X-ORIGIN 'iPlanet Directory
Pro' )
attributeTypes: ( iplanet-am-sample-mail-quota-oid NAME
'iplanet-am-sample-mail-quota' DESC 'iPlanet SampleMailService Attribute'
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 X-ORIGIN 'iPlanet Directory Pro' )
attributeTypes: ( iplanet-am-sample-mail-max-attach-len-oid NAME
'iplanet-am-sample-mail-max-attach-len' DESC 'iPlanet SampleMailService
Attribute' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 X-ORIGIN 'iPlanet Directory
Pro' )
```

Code Example 5-30 sampleMailServiceSchema.ldif File

```
attributeTypes: ( iplanet-am-sample-mail-can-save-address-book-on-server-oid
NAME 'iplanet-am-sample-mail-can-save-address-book-on-server' DESC 'iPlanet
SampleMailService Attribute' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 X-ORIGIN
'iPlanet Directory Pro' )

dn: cn=schema
changetype:modify
add:objectClasses
objectClasses: ( iplanet-am-sample-mail-service-oid NAME
'iplanet-am-sample-mail-service' DESC 'iPlanet dpro SampleMail Service' SUP
top AUXILIARY 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' )
```

sampleMailService.xml File

Following is the sampleMailService.xml file, which is located in the `<dsame_root>/samples/cli/sampleMailService` directory. See the Code Example 5-31 on page 164 for an excerpt from the sampleMailService.xml file which defines some dynamic attributes.

This section gives some information on the `sampleMailService.xml` file. This file can be used as guidelines when creating a service XML file. Refer to the `sampleMailService.xml` file in the sampleMailService directory when reading information in this section.

The "iplanet-am-sample-mail-root-folder" attribute is also specified as a dynamic attribute. The attribute schema is specified with a type of "single", a syntax of "string", and a default value of "Mail", which is the default name for the iplanet-am-sample-mail-root-folder attribute. The i18nKey value is "a2", which maps to "Root Folder", which is the field that displays on the DSAME console for the "iplanet-am-sample-mail-root-folder" attribute. The syntax of "string" means that the user (or administrator) can enter whatever he/she wants to enter in the "Root Folder" field on the sampleMailService service page, in place of the default value of "Mail".

The "iplanet-am-sample-mail-inactivity-interval" attribute is specified as a dynamic attribute. The attribute schema is specified with a type of "single", syntax of "number", i18nKey value of "a6", and a default value of 5. This means that for the "iplanet-am-sample-mail-inactivity-interval" attribute on the sampleMailService service page, the localization field that displays is "Check New Mail Interval (minutes)" will display, and users and administrators will be able to enter a single, number value in this field when creating a modified or new service template.

Several other attributes in the sampleMailService.xml file are specified with attribute types of "single", syntax of "number, and given default values. For example, the "iplanet-am-sample-mail-headers-perpage" attribute is specified as a single type, default value of 10, and given an i18nKey value of "a8", which maps to the "Headers per page" field in the sampleMailService.properties file. This means that for the Headers page page field on the sampleMailService service page in DSAME console, users and administrators can specify any single number in place of the default value given, which is 10.

sampleMailService.properties File

Following is the sampleMailService.properties file, which is located in the `<dsame_root>/samples/cli/sampleMailService` directory.

Code Example 5-31 sampleMailService.properties File

```
#
# PROPRIETARY/CONFIDENTIAL/ Use of this product is subject
# to license terms. Copyright 2001 Sun Microsystems Inc.
# Some preexisting portions Copyright 2001 Netscape
# Communications Corp. All rights reserved.
#

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
```

Every service that you register with DSAME must have a corresponding `.properties` file. All the .properties files for DSAME are located in the following directory:

`<DSAME_root>/web-apps/services/WEB-INF/classes/*.properties`

Each service in DSAME has a corresponding XML (.xml) file and a corresponding .properties file. The XML file defines the attributes for each service. For each attribute in the service's XML file, there is an 'i18nKey" field, for which the value is used as a key to retrieve the actual display message from the service's corresponding .properties file. The DSAME console displays the attributes in alphabetical order according to the i18Nkey definitions in the service XML. For example, if you want two of the attributes to display at the top of the page in DSAME console, give the attributes' i18Nkey definitions values of "a1" and "a2", respectively, and then "b1" and "b2", and so forth, to order the attributes on the Services page. There is a one-to-one mapping between the i18nKey in the XML file and the key in the .properties file.

All the index keys listed in a .properties file (for example, "a1", "p1", etc.) are values for multiple descI18n index keys specified in the service's XML file. When you write a custom service XML and .properties file, you must provide files with all attributes and values that you want to display in DSAME console.

Each service must first be imported with the `amadmin` tool. Then the administrator or user can log into DSAME console and go to the organization he/she wants to register the service for. For each service you add, you must update the DSAME console service hierarchy attribute to get it to display in the service tree. (See Section "Add the sampleMailService to the Service Hierarchy" on page 206 for information on adding a service to the service hierarchy.) At that point, the administrator can log out, log back in, and the service displays for the organization. Then he/she can register and activate the service in DSAME console. (Alternatively, after importing the service, you could use the `registerRequests.xml` to register a service for an existing organization, then activate the service by importing an `activateRequests.xml` to activate the service for an existing organization.)

Below is an excerpt from the sampleMailService.properties file that shows the "a1" and "a2" index key values that are specified in the service's XML file.

Code Example 5-32 Excerpt from sampleMailService.properties File

```
iplanet-am-sample-mail-service-description=Sample Mail Service Profile
a1=Mail Status
a2=Root Folder
```

Every service imported into DSAME using amadmin must have a corresponding and .properties file for it. Also, the relevant .properties filename must be defined in the i18NFileName attribute in the sampleMailService.xml file. In the excerpt from the sampleMailService.xml file below, note that the i18nFileName attribute points to the 'sampleMailService' file.

Code Example 5-33 Excerpt from sampleMailService.xml File

```
ServicesConfiguration>
    <Service name="sampleMailService" version="1.0">
        <Schema
            i18nFileName="sampleMailService"
            i18nKey="iplanet-am-sample-mail-service-description">
    <Global>
            <AttributeSchema name="serviceObjectClasses"
                type="list"
                syntax="string"
                i18nKey="">
                <DefaultValues>
                    <Value>iplanet-am-sample-mail-service</Value>
                </DefaultValues>
            </AttributeSchema>
    </Global>
```

Also, note that each i18nKey for every attribute schema definition in the XML file points to, or is "equal to" a value such as "a1", "a2", or "p1" or "p2" and so forth. These i18nKey values are in the corresponding .properties file for the service (for example, the 'sampleMailService.properties' file for the 'sampleMailService.xml' file). Note that in the sampleMailService.properties file, there is a list of i18nKey fields (also called "localization keys" or "indexes" or "index keys"), which contain the field exactly as it will display on that service page in the DSAME console. For example, using the excerpted sample below, "a1" will display the field "Mail Status" and "a2" displays the field "Root Folder" on the Sample Mail Service Profile page in DSAME console.

Code Example 5-34 Excerpt showing dynamic attribute definitions in sampleMailService.xml File

```
        <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>
```

Code Example 5-34 Excerpt showing dynamic attribute definitions in sampleMailService.xml File

```
                </ChoiceValues>
                <DefaultValues>
                    <Value>Active</Value>
                </DefaultValues>
            </AttributeSchema>
            <AttributeSchema name="iplanet-am-sample-mail-root-folder"
                type="single"
                syntax="string"
                i18nKey="a2">
                <DefaultValues>
                    <Value>Mail</Value>
                </DefaultValues>
            </AttributeSchema>
           <AttributeSchema name="iplanet-am-sample-mail-sentmessages-folder"
                type="single"
                syntax="string"
                i18nKey="a3">
                <DefaultValues>
                    <Value>MailSent</Value>
                </DefaultValues>
            </AttributeSchema>
            <AttributeSchema name="iplanet-am-sample-mail-indent-prefix"
                type="single"
                syntax="string"
                i18nKey="a4">
                <DefaultValues>
                  <Value>|</Value>
                </DefaultValues>
            </AttributeSchema>
            <AttributeSchema name="iplanet-am-sample-mail-initial-headers"
                type="single"
                syntax="number"
                i18nKey="a5">
                <DefaultValues>
                  <Value>10</Value>
                </DefaultValues>
            </AttributeSchema>
           <AttributeSchema name="iplanet-am-sample-mail-inactivity-interval"
                type="single"
                syntax="number"
                i18nKey="a6">
                <DefaultValues>
                  <Value>5</Value>
                </DefaultValues>
            </AttributeSchema>
            <AttributeSchema name="iplanet-am-sample-mail-auto-load"
                type="single"
                syntax="number"
                i18nKey="a7">
                <DefaultValues>
                  <Value>0</Value>
                </DefaultValues>
            </AttributeSchema>
            <AttributeSchema name="iplanet-am-sample-mail-headers-perpage"
                type="single"
```

Code Example 5-34 Excerpt showing dynamic attribute definitions in sampleMailService.xml File

```
        syntax="number"
        i18nKey="a8">
        <DefaultValues>
          <Value>10</Value>
        </DefaultValues>
 </AttributeSchema>
</Dynamic>
```

Explanation of Policy Schema Definitions in sampleMailService.xml

The sampleMailService.xml file gives a brief sample of some policy attributes.

See Section "Overview of Some Policy Concepts and Terms in DSAME" on page 153 and Section "Policy Management Module" on page 153 for information on policy concepts and terms. Section "Attributes and Elements that DSAME Supports" on page 134 for some information of the SMS elements and attributes that use policy (for example, Policy, ActionSchema, ActionValue and ResourceName elements) in the `sms.dtd`.

In Code Example 5-35 on page 169, the Policy element defines a grouping of actions (or privileges) that are specific to the service. Examples of actions are canForwardEmailAddress, canChangeSalaryInformation, etc. These action names define a set of permissions, or privileges; for example,

The ActionSchema element defines a single action (or privilege) for a service. The attribute name in the first policy element definition is:

```
name="iplanet-am-sample-mail-quota"
```

The first ActionSchema element name "iplanet-am-sample-mail-quota" defines a single action or privilege for a service, in this case, it defines a mail quota for the sampleMailService of 1000000 kilobytes, or 1 MB, which is defined to be a type of single choice on the DSAME console, syntax of "number", which means that administrators or users must enter a number when creating a new service template, and the default value given for administrators is 1000000 (KB). The `i18nKey="p1"` definition means that the localization, or index, key points to the "p1" index or key in the sampleMailService.properties file, which maps to the field Mail Quota, which will display for the iplanet-am-sample-mail-quota attribute on the DSAME console services page.

The next ActionSchema element definition in the `sampleMailService.xml` file is:

```
ActionSchema name="iplanet-am-sample-mail-max-attach-len"
```

Similarly, the "iplanet-am-sample-mail-quota" Action name defines a set of permissions or privileges, which will appear as "single" type choice on the service page in DSAME console, which means that administrators and users can specify a single type value, which must have be a number. The i18nKey value maps to "p2", which maps to the Auto-download Maximum Attachment Length field, which is the exact field that will display for the "ipalent-am-sample-mail-max-attach-len" atttribute on the sampleMailService services page in DSAME console

```
p2=Auto-download Maximum Attachment Length
```

The "iplanet-am-sample-mail-max-attach-len" attribute is given a default value of 100000 which will display on the sampleMailServices page in DSAME console, which administrators can change, provided the value they enter matches the ActionSchema definition.

NOTE DSAME 5.0 does not support default values for policy schema definitions.

Code Example 5-35 Excerpt from sampleMailService.xml defining Policy Schema

```
            <Policy>
                <ActionSchema name="iplanet-am-sample-mail-quota"
                    type="single"
                    syntax="number"
                    i18nKey="p1">
                </ActionSchema>
                <ActionSchema name="iplanet-am-sample-mail-max-attach-len"
                    type="single"
                    syntax="number"
                    i18nKey="p2">
                </ActionSchema>
                <ActionSchema
name="iplanet-am-sample-mail-can-save-address-book-on-server"
                    type="single"
                    syntax="boolean"
                    i18nKey="p3">
                </ActionSchema>
            </Policy>
        </Schema>
    </Service>
</ServicesConfiguration>
```

Policy Schema must be defined before Policy Template can be Created in DSAME Console

It is important to note that you must add services (import the service using the `amadmin` tool) that have policy subschema defined for the organization before you can create a policy template. Once the service with policy schema defined has been imported into DSAME, then you can create a policy template within DSAME console.

If the DSAME console detects that an organization does not have any services defined for that organization, it will not create a policy template, and will give an error.

amAdmin.dtd Used when Performing Batch Updates to DIT

The amAdmin.dtd is the DTD that is used by DSAME when you perform batch operations on the DIT (such as creations, deletions, gets on user objects such as roles, organizations, users, people containers, and groups). When writing XML files that perform batch operations on the DIT, administrators or customization engineers must follow the document structure defined in this DTD.

NOTE The XML files that define services use a different DTD—the `sms.dtd`. The XML files that perform batch updates on the Directory server use the `amAdmin.dtd`.

Batch Operations you can perform using the amAdmin.dtd

The purpose of the `amAdmin.dtd` is to enable you to perform the following command line operations on DSAME. Using the `amadmin` tool with the --data option enables an administrator or customization engineer to:

- Create roles, users, organizations, groups, people containers and services.
- Create users, suborganizations, roles, groups, sub-containers, people containers, sub-people containers, sub-groups.

- Delete roles, users, organizations, groups, people containers and services.
- 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 services.
- Register a service for an existing organization.
- Unregister a service from an existing organization.
- Get registered service names from an existing organization.
- Get activated service names from an existing organization.
- Get the total number of registered and activated services for an existing organization.
- Execute requests in multiple XML files together on the amadmin command line. (The order is important that the batch update XML files are imported is important. For example, you must register a service before it can be activated; similarly, a service must be deactivated before it can be unregistered.)

Files Used to perform Batch Updates to DIT

There are several sample files provided in the following directory, which enable you to perform the operations described in Section "Batch Operations you can perform using the amAdmin.dtd" on page 170:

```
<dsame_root>/samples/admin/cli/bulk-ops
```

The sample files provided for you to perform batch operations are:

- createRequests.xml
- activateRequests.xml
- deactivateRequests.xml
- deleteGroupRequests.xml
- deleteOrgRequests.xml
- deletePCRequests.xml
- deletePolicyRequests.xml

- getActivatedServices.xml
- getNumOfServices.xml
- getRegisteredServices.xml
- getRequests.xml
- registerRequests.xml
- unregisterRequests.xml

See the Section "Sample File (createRequests.xml) to Perform batch Updates to DIT" on page 188 for a brief explanation of the createRequests.xml file, and refer to Chapter 6, "Using the Command Line Interface for information on how to import batch operations sample files into the DIT.

Description of amAdmin.dtd

This section provides some information on the amAdmin.dtd which defines some rules that must be adhered to, when writing and using batch operations XML files in DSAME.

The amAdmin.dtd uses standard XML (Extensible Markup Language) syntax, elements, and attributes. The amAdmin.dtd has been written specifically for use with the DSAME product, and the attributes are specific to DSAME and Directory server. For explanations of standard XML elements, syntax, and attributes, refer to any of the numerous XML books or web sites available, or go to the following URL:

```
http://xml101.com
```

The `amAdmin.dtd` reflects the structure of the DSAME SDK. The `amAdmin.dtd` follows a request-based paradigm. Neither the amAdmin.dtd nor the `amadmin` CLI tool are meant to fully duplicate the DSAME console (GUI) functionality available through the `amadmin` CLI tool.

The following sections provide brief explanations of the batch update tasks you can perform on the DIT, such as creating, deleting, and reading roles, users, organizations, groups, people containers, and services from the Directory server tree (DIT). The administrator or customization engineer must configure some sample batch update XML files per the rules defined in the `amAdmin.dtd`. Some sample batch operations XML files are located in the following directory:

```
<dsame_root>/samples/admin/cli/bulk-ops
```

Note that there are descriptions of the various `amAdmin.dtd` elements and attributes used when performing batch operations on the DIT is provided in the `amAdmin.dtd` file itself.

NOTE You should not modify the `amAdmin.dtd` in any way. It contains rules and definitions that control the way batch operations are performed on the Directory Information Tree (DIT), and must not be modified; otherwise, errors could be introduced into your Directory server DIT, or the batch operations will not work correctly.

Requests Element

The Requests element is the root element of the input XML document to `amadmin`. It must contain at least on child element. The child elements are designe dto follow an object-oriented model where the actual requests are performed on the high level DSAME objects such as Organization, Container, People Container, Role, and Group. To enable batch processing, the root element can accept more than one set of requests.

The Requests element defines all the directory objects that can be performed on the Directory server DIT:

- Organization
- Container
- People Container
- Role
- Group

See Code Example 5-36 on page 173 for a sample Requests Element and Requests sub-element.

Code Example 5-36 Requests Element and Requests Sub-Elements

```
<!ELEMENT Requests (
    OrganizationRequests |
    ContainerRequests |
    PeopleContainerRequests |
    RoleRequests |
    GroupRequests )+
>
```

OrganizationRequests Element

The OrganizationRequests element combines all the requests that are to be performed on Organization type objects. To enable batch processing, this element can have one or more child elements. The child elements, as suggested by their intuitive names, represent the various requests. Note that all the child elements perform their operations on the same instance of Organization object. If you want to manipulate different Organization objects, you can include different OrganizationRequests elements in the root element Requests.

All the organization requests are shown in the on page 174 (for example, create suborganizations, get roles, unregsiter services, delete sub-organizations, etc.)

Code Example 5-37 OrganizationRequests Element

```
<!ELEMENT OrganizationRequests (
    (CreateSubOrganization)*,
    (CreatePeopleContainer)*,
    (CreateRole)*,
    (CreateGroup)*,
    (CreatePolicy)?,
    (GetSubOrganizations)?,
    (GetPeopleContainers)?,
    (GetRoles)?,
    (GetGroups)?,
    (GetUsers)?,
    (RegisterServices)?,
    (UnregisterServices)?,
    (ActivateServices)?,
    (DeactivateServices)?,
    (GetActivatedServiceNames)?,
    (GetRegisteredServiceNames)?,
    (GetNumberOfServices)?,
    (DeleteRoles)?,
    (DeleteGroups)?,
    (DeletePolicy)?,
    (DeletePeopleContainers)?,
    (DeleteSubOrganizations)? )
>
```

The DN attribute of the element OrganizationRequests specifies the DN (Distinguished Name, for example, "uid=amAdmin,ou=People,o=iplanet.com,o=isp") of the Organization element, on which all of the requests (specified by the child elements) will be performed.

Note that the DN attribute is a required attribute; the service developer or customization engineer must supply a DN on the `amadmin` command line when performing the batch operation.

Code Example 5-38 DN Attribute of OrganizationRequests Element

```
<!-- The DN attribute of the element, OrganizationRequests, specifies
the DN of the Organization object on which all of the requests
(specified by the child elements) shall be made.
-->

<!ATTLIST OrganizationRequests
    DN    CDATA    #REQUIRED
>
```

CreateSubOrganization Element

The CreateSubOrganization element can have zero or more AttributeValuePairs.

The Attribute element must be all one word without spaces, thus, it is specified as NMTOKEN in the amAdmin.dtd. (See on page 175.)

Code Example 5-39 CreateSubOrganization Element

```
<!-- Note that CreateSubOrganization element can have zero or more
AttributeValuePairs.
-->

<!ELEMENT CreateSubOrganization    (AttributeValuePair)* >
<!ATTLIST CreateSubOrganization
    createDN    CDATA    #REQUIRED
>
<!ELEMENT AttributeValuePair (Attribute, (Value)+) >
<!ELEMENT Attribute EMPTY >

<!-- Attribute must be all one word without spaces. Hence, it is
indicated as NMTOKEN below.
-->

<!ATTLIST Attribute
    name    NMTOKEN        #REQUIRED
>
<!ELEMENT Value (#PCDATA) >
```

CreatePeopleContainer Element

The CreatePeopleContainer element lets you create an attribute / value pair on the specified people container in the DIT. You must specify a "createDN" attribute.

CreateGroup Element

The CreateGroup Element lets you create a specified group in the DIT. You must specify a "createDN" attribute.

CreateRole Element

The CreateRole element lets you create a specified role in the DIT, and specify attribute/value pairs also. You must specify a "createDN" attribute. (See on page 176.)

Code Example 5-40 CreatePeopleContainer, CreateGroup, and CreateRole Elements

```
<!ELEMENT CreatePeopleContainer (AttributeValuePair)* >
<!ATTLIST CreatePeopleContainer
    createDN    CDATA    #REQUIRED
>

<!-- _____________________________________________________________________  -->

<!ELEMENT CreateGroup EMPTY >
<!ATTLIST CreateGroup
    createDN    CDATA    #REQUIRED
>

<!-- _____________________________________________________________________  -->

<!ELEMENT CreateRole (AttributeValuePair)* >
<!ATTLIST CreateRole
    createDN    CDATA    #REQUIRED
>

<!-- _____________________________________________________________________  -->
```

CreatePolicy Element

The CreatePolicy element lets you create one or more policy objects in the DIT. Policy is the root element that defines a named policy. The attribute "name" specifies the policy name. "serviceName" attribute identifies the service name to which the named policy applies.

The Policy element can have one or more rules specified for it (specified by Rule+).

Code Example 5-41 Create Policy Element

```
<!ELEMENT CreatePolicy (Policy)+ >
<!ATTLIST CreatePolicy
    createDN    CDATA    #REQUIRED
>

<!-- Policy is the root element that defines a named policy.
The attribute "name" provides the policy name.
"serviceName" identifies the service name for which the policy is
applicable.
The policy element can have one or more rules (Rule).
-->

<!ELEMENT       Policy          ( Rule+ ) >
<!ATTLIST       Policy
        name            CDATA           #REQUIRED
        serviceName     NMTOKEN         #IMPLIED
>
```

Rule Element with ServiceName, ResourceName?, and AttributeValuePair+ Sub-Elements

The Rule element defines a policy rule that contains a service name, resource name, and its action names and corresponding values. The "name" attribute provides a user-friendly name for the rule. The sub-element "ServiceName" gives the name of the service; the sub-element "ResourceName" gives the resource name, and the "AttributeValuePair" provides the action names and corresponding action values.

The sub-element "ServiceName" provides the name of the service for which a rule has been created. The attribute "name" provides the service name.

The sub-element "ResourceName" provides the name of the reource for which a rule has been created. If the service does not have a resource, this element will not be there for the rule. The "name" attribute provides the resource name. (See on page 177.)

Code Example 5-42 Rule Element with Sub-Elements

```
<!ELEMENT       Rule            ( ServiceName, ResourceName?,
                                  AttributeValuePair+ ) >
<!ATTLIST       Rule
        name            CDATA           #REQUIRED
>

<!-- ServiceName provides the name of the service for which a
rule has been created. The attribute "name" provides the service
name.
```

Code Example 5-42 Rule Element with Sub-Elements

```
-->

<!ELEMENT       ServiceName             EMPTY>
<!ATTLIST       ServiceName
        name            CDATA           #REQUIRED
>

<!-- ResourceName provides the name of the resource for which
a rule has been created. If the service does not have resource,
this element will not be there for the rule. The attribute "name"
provides the resource name.
-->

<!ELEMENT       ResourceName            EMPTY>
<!ATTLIST       ResourceName
        name            CDATA           #REQUIRED
>
```

GetSubOrganizations Element

For the GetSubOrganizations element, for those objects that have LDAP attributes, all get operations in the `amAdmin.dtd` follow the same design pattern. If the element has an XML attribute "DNsOnly" set to true, or does not have the "DNsOnly" attribute, only the DNs of the corresponding DSAME objects will be returned. If DNsOnly="false", the entire object (with the LDAP attribute value pairs) will be returned. However, the behavior of DNsOnly is valid ONLY if there are no child elements (DNs) specified; if the DNs are specified, the entire object will always be returned. (See on page 178.)

Code Example 5-43 GetSubOrganizations Element

```
<!ELEMENT GetSubOrganizations (DN)* >

<!-- For those objects that may have LDAP attributes, all get operations in
this DTD follow the same design pattern: If the element has an XML
attribute DNsOnly set to true or doesn't have that XML attribute, only
the DNs of the corresponding DSAME objects shall be
returned. If DNsOnly="false", the entire object (with the LDAP
attribute value pairs) shall be returned.
However, the behavior of DNsOnly is valid ONLY if there are no child
elements (DNs) specified; if the DNs are specified, the entire
object shall always be returned.
-->

<!ATTLIST GetSubOrganizations
```

Code Example 5-43 GetSubOrganizations Element

```
    DNsOnly    (true | false) "true"
>
```

GetPeopleContainers, GetGroups, and GetRoles Elements

For the GetPeopleContainers element, if there are no child elements (no DNs) specified, then ALL PeopleContainers at all levels within the specified Organization will be returned.

For the GetGroups element, you must specify a "level" attribute, or node, for which you want to get the groups returned. You can specify "ONE_LEVEL" to get just the groups at that node level, or you can specify a "SUB-TREE" attribute, which gets all the groups for at that node's level and all the nodes underneath that node.

For the GetRoles element, you must specify a "level" attribute, or node, for which you want to get the roles returned. You can specify "ONE_LEVEL" to get just the roles at that node level, or you can specify a "SUB-TREE" attribute, which gets all the roles for at that node's level and all the nodes underneath that node.

Code Example 5-44 GetPeopleContainers, GetGroups, and GetRoles Elements

```
<!ELEMENT GetPeopleContainers (DN)* >
<!ATTLIST GetPeopleContainers
    DNsOnly    (true | false) "true"
>

<!-- _____________________________________________________________________  -->

<!ELEMENT GetGroups EMPTY >
<!ATTLIST GetGroups
    level    (ONE_LEVEL | SUB_TREE) "SUB_TREE"
>

<!-- _____________________________________________________________________  -->

<!ELEMENT GetRoles EMPTY >
<!ATTLIST GetRoles
    level    (ONE_LEVEL | SUB_TREE) "SUB_TREE"
>
```

GetUsers Element

For the GetUsers element, if there are no child elements (no DNs) specified, then ALL users at all levels within this object are returned. You must specify a true or false value for the GetUser element; the default is true.

Code Example 5-45 GetUsers Element

```
<!ELEMENT GetUsers (DN)* >
<!ATTLIST GetUsers
    DNsOnly    (true | false) "true"
>
```

RegisterServices and UnregisterServices Elements

For the RegisterServices element, the service schema for the service must have been loaded by using the amadmin CLI tool before a service can be registered. Multiple services can be registered using this tag. One or more Service_Name sub-elements can be specified on which to register services.

For the UnregisterServices element, it can be ued to unregister a previously-registered service. If the service was not previously registered, the request is simply ignored for that service. Multiple services can be unregistered using this tag. If no Service_Name tag is specified, then no registered service will be unregistered, the request is simply ignored.

Code Example 5-46 RegisterServices and UnregisterServices Elements

```
<!-- Before a service is registered, the service schema for the service must
have been loaded using the CLI. Multiple services can be registered using this
tag.
-->

<!ELEMENT RegisterServices (Service_Name)+ >
<!ELEMENT Service_Name (#PCDATA) >

<!-- ____________________________________________________________________  -->

<!-- Unregister a previously registered service. If the service was not
previously registered, the request is simply ignored for that service. Multiple
services can be unregistered using this tag. If no Service_Name tag is
specified, then no registered service will be unregistered, the request is
simply ignored.
-->

<!ELEMENT UnregisterServices (Service_Name)* >
```

ActivateServices and DeactivateServices Elements

For the ActivateServices element, a service must have been registered before it can be activated. Before a service is registered, the service schema must have been loaded using the amadmin CLI tool. If no Service_Name tag is specified, then no registered service will be activated, the request is simply ignored.

For the DeactivateServices element, a service must have been registered and activated before it can be deactivated. Before a service is registered, the service schema must have been loaded using the amadmin CLI tool. If the service was not previously registered and/or activated, the request is ignored for that service. Multiple services can be deactivated using this tag. If no Service_Name tag is specified, then no registered service will be deactivated, the request is simply ignored.

Code Example 5-47 ActivateServices and DeactivateServices Elements

```
<!ELEMENT ActivateServices (Service_Name)* >

<!-- _____________________________________________________________________  -->

<!-- A service must have been registered and activated before it can be
deactivated. Before a service is registered, the service schema must have been
loaded using the CLI. If the service was not previously registered and/or
activated, the request is simply ignored for that service. Multiple services
can be deactivated using this tag. If no Service_Name tag is specified, then
no registered service will be deactivated, the request is simply ignored.
-->

<!ELEMENT DeactivateServices (Service_Name)* >

<!-- _____________________________________________________________________  -->

<!-- ALL activated services within this object are returned.
-->
```

GetActivatedServiceNames, GetRegisteredServiceNames, and GetNumberofServices Elements

For the GetActivatedServiceNames element, all activated services within this object are returned.

For the GetRegisteredServiceNames element, all registered services within this object are returned.

For the GetNumberOfServices element, the total number of services within this object are returned.

Code Example 5-48 GetActivatedServiceNames, GetRegisteredServicesNames, and Get GetNumberOfServices Elements

```
<!-- ALL activated services within this object are returned.
-->

<!ELEMENT GetActivatedServiceNames EMPTY >

<!-- _____________________________________________________________________  -->

<!-- ALL registered services within this object are returned.
-->

<!ELEMENT GetRegisteredServiceNames EMPTY >

<!-- _____________________________________________________________________  -->


<!-- Total number of services within this object are returned.
-->

<!ELEMENT GetNumberOfServices EMPTY >
```

DeleteSubOrganizations Element

For those objects that have children objects (such as containers, organizations, people containers), all delete operations in the amAdmin.dtd follow the same design pattern: If deleteRecursively=”false”, then accidental deleting entire subtrees is avoided. It could be disastrous if deleteRecursively=”true”. To avoid accidental and unintentional deletions, the default value is “false”.

Code Example 5-49 DeleteSubOrganizations Element

```
<!ELEMENT DeleteSubOrganizations (DN)+ >
<!ATTLIST DeleteSubOrganizations
    deleteRecursively (true | false) "false"
>
```

DeletePeopleContainers Element

For those objects that have children objects (such as containers, organizations, people containers), all delete operations in this DTD follow the same design pattern: If deleteRecursively=”false”, then accidentally deleting entire subtrees is avoided.

Code Example 5-50 DeletePeopleContainers Element

```
<!ELEMENT DeletePeopleContainers (DN)+ >
<!ATTLIST DeletePeopleContainers
    deleteRecursively (true | false) "false"
>
```

DeleteGroups Element

For those objects that might have children objects (such as containers, organizations, people containers), all delete operations in this DTD follow the same design pattern: If deleteRecursively=”false”, then accidentally deleting entire subtrees is avoided. It can be disastrous if deleteRecursively=”true”. The default value is ‘false’ to avoid accidental and unintentional deletions.

Code Example 5-51 DeleteGroups Element

```
<!ELEMENT DeleteGroups (DN)+ >
<!ATTLIST DeleteGroups
    deleteRecursively (true | false) "false"
>
```

DeleteRoles Element

The DeleteRoles element deletes roles, based on the DN.

Code Example 5-52 DeleteRoles element

```
<!ELEMENT DeleteRoles (DN)+ >
```

DeletePolicy Element

The DeletePolicy element deletes the policy applicable to a service, given the DN.

Code Example 5-53 DeletePolicy Element

```
<!ELEMENT DeletePolicy (PolicyName)+ >
<!ATTLIST DeletePolicy
    deleteDN    CDATA    #REQUIRED
>
```

PolicyName Element

The PolicyName element gives the name of the policy applicable to a service.

Code Example 5-54 PolicyName Element

```
<!ELEMENT       PolicyName              EMPTY>
<!ATTLIST       PolicyName
        name            CDATA           #REQUIRED
>
```

ContainerRequests Element

The ContainerRequests element aggregates all the requests that are to be performed on Container type objects. To enable batch processing, this element can have one or more child elements. The child elements, as suggested by their intuitive names, represent the various requests. Note that all the child elements operate on the same instance of Container object. If different Container objects are to be manipulated, different ContainerRequests elements can be included in the root element Requests.

See Code Example 5-55 on page 185 for a list of the types of container requests you can perform on the DIT in a batch operation; for exame, you can create roles, groups, policies, activate and deactivate services, delete roles and groups, and other requests on objects in the DIT.

Note that you can when you create roles in the DIT, you can specify attribute/value pairs. You can create sub-containers with attribute/value pairs, create sub-people-containers with attribute/value pairs, and create users with attribute/value pairs.

Code Example 5-55 ContainerRequests Element

```
<!ELEMENT ContainerRequests (
    (CreateSubContainer)*,
    (CreatePeopleContainer)*,
    (CreateRole)*,
    (CreateGroup)*,
    (CreatePolicy)?,
    (GetSubContainers)?,
    (GetPeopleContainers)?,
    (GetRoles)?,
    (GetGroups)?,
    (GetUsers)?,
    (RegisterServices)?,
    (UnregisterServices)?,
    (ActivateServices)?,
    (DeactivateServices)?,
    (GetActivatedServiceNames)?,
    (GetRegisteredServiceNames)?,
    (GetNumberOfServices)?,
    (DeleteRoles)?,
    (DeleteGroups)?,
    (DeletePolicy)?,
    (DeletePeopleContainers)?,
    (DeleteSubContainers)? )
>

<!-- The DN attribute of the element, ContainerRequests, specifies
the DN of the Container object on which all of the requests
(specified by the child elements) shall be made.
-->

<!ATTLIST ContainerRequests
    DN    CDATA    #REQUIRED
>

<!-- ____________________________________________________________________  -->

<!-- Note that CreateSubContainer element can have zero or more
AttributeValuePairs.
-->

<!ELEMENT CreateSubContainer    (AttributeValuePair)* >
<!ATTLIST CreateSubContainer
    createDN    CDATA    #REQUIRED
>

<!-- ____________________________________________________________________  -->

<!-- If there are no child elements (no DNs) specified, then ALL
subContainers at all levels within this Container are returned.
-->

<!ELEMENT GetSubContainers (DN)* >

<!-- For those objects that may have LDAP attributes, all get operations in
```

Code Example 5-55 ContainerRequests Element

```
this DTD follow the same design pattern: If the element has an XML
attribute DNsOnly set to true or doesn't have that XML attribute, only
the DNs of the corresponding DSAME objects shall be
returned. If DNsOnly="false", the entire object (with the LDAP
attribute value pairs) shall be returned.
However, the behavior of DNsOnly is valid ONLY if there are no child
elements (DNs) specified; if the DNs are specified, the entire
object shall always be returned.
-->

<!ATTLIST GetSubContainers
    DNsOnly    (true | false) "true"
>

<!-- _____________________________________________________________________  -->

<!-- For those objects that may have children objects (such as containers,
organizations, people containers), all delete operations
in this DTD follow the same design pattern: If deleteRecursively="false",
then accidental deletion of entire subtrees is avoided.
It can be disastrous if deleteRecursively="true". The default value is 'false'
to avoid accidental and unintentional deletions.
-->

<!ELEMENT DeleteSubContainers (DN)+ >
<!ATTLIST DeleteSubContainers
    deleteRecursively (true | false) "false"
>

<!-- ======================== PeopleContainerRequests =================== -->

<!ELEMENT PeopleContainerRequests (
    (CreateSubPeopleContainer)*,
    (CreateUser)*,
    (GetNumberOfUsers)?,
    (GetUsers)?,
    (GetSubPeopleContainers)?,
    (DeleteUsers)?,
    (DeleteSubPeopleContainers)? )
>

<!-- The DN attribute of the element, PeopleContainerRequests, specifies
the DN of the PeopleContainerRequests object on which all of the requests
(specified by the child elements) shall be made.
-->

<!ATTLIST PeopleContainerRequests
    DN    CDATA    #REQUIRED
>

<!ELEMENT CreateSubPeopleContainer (AttributeValuePair)* >
<!ATTLIST CreateSubPeopleContainer
    createDN    CDATA    #REQUIRED
>
```

Code Example 5-55 ContainerRequests Element

```
<!ELEMENT CreateUser (AttributeValuePair)* >
<!ATTLIST CreateUser
    createDN    CDATA    #REQUIRED
>

<!-- For those objects that may have children objects (such as containers,
organizations, people containers), all delete operations
in this DTD follow the same design pattern: If deleteRecursively="false",
then accidental deletion of entire subtrees is avoided.
It can be disastrous if deleteRecursively="true". The default value is 'false'
to avoid accidental and unintentional deletions.
-->

<!ELEMENT DeleteSubPeopleContainers (DN)+ >
<!ATTLIST DeleteSubPeopleContainers
    deleteRecursively (true | false) "false"
>

<!ELEMENT DeleteUsers (DN)+ >

<!ELEMENT GetSubPeopleContainers (DN)* >
<!ATTLIST GetSubPeopleContainers
    level    (ONE_LEVEL | SUB_TREE) "SUB_TREE"
    DNsOnly    (true | false) "true"
>

<!-- ======================== RoleRequests =================== -->

<!ELEMENT RoleRequests (
    (GetNumberOfUsers)?,
    (GetUsers)?,
    (AddUsers)?
    )
>

<!-- The DN attribute of the element, RoleRequests, specifies
the DN of the RoleRequests object on which all of the requests
(specified by the child elements) shall be made.
-->

<!ATTLIST RoleRequests
    DN    CDATA    #REQUIRED
>

<!ELEMENT GetNumberOfUsers EMPTY >

<!ELEMENT AddUsers (DN)+ >
<!ELEMENT DN (#PCDATA) >

<!-- ======================== GroupRequests =================== -->

<!ELEMENT GroupRequests (
    (CreateSubGroup)*,
    (GetSubGroups)?,
    (GetNumberOfUsers)?,
```

Code Example 5-55 ContainerRequests Element

```
    (GetUsers)?,
    (AddUsers)?,
    (DeleteSubGroups)? )
>

<!-- The DN attribute of the element, GroupRequests, specifies
the DN of the GroupRequests object on which all of the requests
(specified by the child elements) shall be made.
-->

<!ATTLIST GroupRequests
    DN    CDATA    #REQUIRED
>

<!ELEMENT GetSubGroups (DN)* >
<!ATTLIST GetSubGroups
    level    (ONE_LEVEL | SUB_TREE) "SUB_TREE"
    DNsOnly    (true | false) "true"
>

<!ELEMENT CreateSubGroup (AttributeValuePair)* >
<!ATTLIST CreateSubGroup
    createDN    CDATA    #REQUIRED
>

<!-- For those objects that may have children objects (such as containers,
organizations, people containers), all delete operations
in this DTD follow the same design pattern: If deleteRecursively="false",
then accidental deletion of entire subtrees is avoided.
It can be disastrous if deleteRecursively="true". The default value is 'false'
to avoid accidental and unintentional deletions.
-->

<!ELEMENT DeleteSubGroups (DN)+ >
<!ATTLIST DeleteSubGroups
    deleteRecursively (true | false) "false"
>
```

Sample File (createRequests.xml) to Perform batch Updates to DIT

Below is one of the sample XML files provided in the `<dsame_install_dir/SUNWam/samples/admin/cli/bulk-ops` directory. For example, you would use the createRequests.xml file to create various objects such as users, roles, people containers, groups, etc. in the DIT.

Code Example 5-56 createRequests.xml File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Requests
    PUBLIC "-//iPlanet//DSAME 5.0 Admin CLI DTD//EN"
    "file:/opt/SUNWam/web-apps/services/dtd/amAdmin.dtd"
>

<!--  CREATE REQUESTS -->

<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"/>

    <CreatePolicy createDN="o=sun.com,o=isp">
        <Policy name="ManagerPolicy"
             serviceName="sampleMailService">
             <Rule name="Manager Rule">
                 <ServiceName name="sampleMailService"/>
                 <ResourceName name="sampleMailService"/>
     <AttributeValuePair>
             <Attribute name="iplanet-am-sample-mail-quota"/>
             <Value>1000000</Value>
     </AttributeValuePair>
     <AttributeValuePair>
             <Attribute name="iplanet-am-sample-mail-max-attach-len"/>
             <Value>100000</Value>
     </AttributeValuePair>
     <AttributeValuePair>
             <Attribute
name="iplanet-am-sample-mail-can-save-address-book-on-server"/>
             <Value>false</Value>
     </AttributeValuePair>
              </Rule>
        </Policy>
    </CreatePolicy>



</OrganizationRequests>

<PeopleContainerRequests DN="ou=People1,o=sun.com,o=isp">

    <CreateSubPeopleContainer createDN="SubPeople1">
    <AttributeValuePair>
```

Code Example 5-56 createRequests.xml File *(Continued)*

```
        <Attribute name="description"/>
        <Value>SubPeople description</Value>
    </AttributeValuePair>
    </CreateSubPeopleContainer>

    <CreateUser createDN="amUser">
       <AttributeValuePair>
          <Attribute name="cn"/>
          <Value>amUser</Value>
       </AttributeValuePair>
       <AttributeValuePair>
          <Attribute name="sn"/>
          <Value>amUser </Value>
       </AttributeValuePair>
       <AttributeValuePair>
          <Attribute name="userPassword"/>
          <Value>12345678</Value>
       </AttributeValuePair>
    </CreateUser>

</PeopleContainerRequests>

<RoleRequests DN="cn=ManagerRole,o=sun.com,o=isp">

    <AddUsers>
        <DN>uid=amUser,ou=People1,o=sun.com,o=isp</DN>
    </AddUsers>

</RoleRequests>

<GroupRequests DN="cn=ContractorsGroup,o=sun.com,o=isp">

    <CreateSubGroup createDN="SubContractorsGroup">
    <AttributeValuePair>
        <Attribute name="uniquemember"/>
        <Value>uid=amUser,ou=People1,o=sun.com,o=isp</Value>
    </AttributeValuePair>
    </CreateSubGroup>

    <AddUsers>
        <DN>uid=amUser,ou=People1,o=sun.com,o=isp</DN>
    </AddUsers>


</GroupRequests>
```

Using the Command Line Interface

This chapter describes how you can use the `amadmin` command line interface tool. It contains the following sections:

- Overview of the amadmin Command Line Interface Tool
- How the amadmin CLI Tool Works
- Guidelines for Loading Services into DSAME
- Guidelines on Performing Batch Updates to User Objects in Directory Server
- Steps to Perform Batch Updates to DIT
- Tips when running amadmin Tool

Overview of the amadmin Command Line Interface Tool

This chapter describes the command-line interface (`amadmin`) available for iPlanet Directory Server Access Management Edition administration. The primary purpose of the amadmin tool is to help an administrator perform batch administrative tasks on the Directory Server, for example, create, register, and activate new services; and create, delete, and read (get) organizations, people containers, groups, roles, and users.

The primary purposes of the amadmin CLI tool in this release are to:

- Load service schema definitions in XML files. This lets DSAME administrators load their services into DSAME (and iPlanet Directory Server) in XML file format using the `amadmin` CLI tool, and then implement the service based on the schema. In this release of DSAME, you MUST load all services into DSAME (and Directory Server) by using the amadmin tool; you cannot load or import

services through DSAME's Administration Console.

To register a service through the amadmin CLI tool, you first define the service schema and configuration data in XML file(s), and then import the XML file(s) using the amadmin CLI tool.

- Perform batch operations (such as creating, deleting, and reading user objects in the Directory server, such as users, organizations, groups, people containers, roles, etc.) For example, if an administrator wants to create 1000 organizations, people containers, users, and groups, he/she can do this in one attempt by putting all the requests in one or multiple XML files for performing batch operations (such as `<dsame_root>/samples/admin/cli/bulk-ops/createRequests.xml` and `<dsame_root>/samples/admin/cli/bulk-ops/deleteRequests.xml` files) and run the amadmin tool once. Otherwise, an administrator has to create these objects in Directory Server one at a time in DSAME Administration Console.

The amadmin CLI tool takes one or more XML files as input. Then it reads all the requests described in the XML file(s) and submits these requests to Directory Server through the DSAME SDK. Such requests include those for creating, getting and deleting various directory objects, as mentioned above.

There are two DTDs that the XML files should adhere to:

`sms.dtd`—All the service XML files (for example, all the service XML files located in the `<dsame_root>/SUNWam/web-apps/services/dtd` directory) must adhere to the Service Management Services) sms.dtd file.

`amAdmin.dtd`—All the XML files that perform batch updates to the Directory Information Tree (or DIT), which are located in the `<dsame_root>/web-apps/services/amAdmin.dtd` directory, must adhere to the document type definition rules defined in the `amAdmin.dtd` file.

In this chapter, there are some sample XML files (one for loading a service, and one for performing a batch load operation such as a create requests operation to create some user objects in the DIT, two DTDs (an sms.dtd and an amAdmin.dtd), and how to load the service into DSAME. The sample XML files provided with this build are located in:

The DTD files (`sms.dtd` and `amAdmin.dtd`) are located in:

`<dsame_root>/SUNWam/web-apps/services/dtd`

They set up rules for how the XML files should be written, how an administrator would perform batch operations on the DIT, for example, adding new users and roles to the Directory server in batch mode. An administrator may want to perform a large number of operations on the DIT, and could use the sample XML files provided to perform creates, deletes, and gets on the Directory server.

NOTE When loading services and when performing batch updates on the Directory server, two different dtds are used; services use the `sms.dtd`; batch updates to the DIT use the `amAdmin.dtd`.

The `amadmin` CLI tool is not intended to be a replacement for Admin Console Tool, which has a GUI interface. The amadmin CLI tool does not support all the features that can be executed from DSAME's Admin Console. It is recommended that you use Admin Console GUI for any incremental administrative tasks while you use the `amadmin` CLI tool for any batch administrative tasks.

How the amadmin CLI Tool Works

The primary function of amadmin CLI tool comprises two different phases:

Service schema definition in XML and registration

To define the metadata (what type of attributes the service will have, how many attributes the service will have, what kind of policy attributes the service will have, what the attribute names are, etc.)

Data creation in Directory Server DIT (or populating the Directory Server DIT):

To actually create the DIT entries such as organization, group, people container, user, and role, the attributes defined in these entries will use the schema defined in the service XML file. For example, if an attribute A in the "user" entry is from some service X, the type (metadata/schema) of the attribute (string, boolean, integer, or some other attribute type) is determined using the schema defined in the service schema definition in the service XML file (for example, amAdminConsole.xml).

What you can use the amadmin tool for

The `amadmin` CLI tool provides a way for you to perform the following actions on the Directory server:

1. Import services.
2. Create roles, users, organizations, groups, people containers, and services.
3. Delete roles, users, organizations, groups, people containers, and services.
4. Read (get) roles, users, organizations, groups, people containers, and services.
5. Read (get) roles, people containers, users.
6. Read (get) number of users for groups, people containers, roles.
7. Register a service for an existing organization.
8. Unregister a service from an existing organization.
9. Activate a service for an existing organization.
10. Deactivate a service from an existing organization.
11. Get registered service names from an existing organization.
12. Get activated service names from an existing organization.
13. Get the total number of registered and activated services for an existing organization.
14. Execute requests in multiple XML files together.

File pathnames follow UNIX filesystem conventions. `<dsame_root>` refers to the installation directory where the Directory Server (iPlanet Directory Server) is installed, and is `/usr/iplanet/servers` by default.
`<dsame_root>` refers to the installation directory where DSAME services and agents are installed and is `/opt/SUNWam` by default.

Requirements to run amadmin CLI Tool

To perform batch updates on the Directory Server (DIT), you will need a sample XML file conforming to the DTD. This DTD is located in

`<dsame_root>/web-apps/services/dtd/amAdmin.dtd`

To load services, you will need to use one of the sample XML files provided for defining service schema and configuration data, or you can use one of these to create your own service XML. Refer to the DTD provided for when loading services. This DTD is located in

<dsame_root>/web-apps/services/dtd/sms.dtd

The <dsame_root> by default is /opt/SUNWam. If you install DSAME using the default DSAME installation directory, this is where the DSAME build will be installed.

Installation/Setup

When DSAME 5.0 is installed, the amAdmin CLI tool also gets installed along with the server. The amserver is located in <dsame_root>/web-apps/services/WEB-INF/bin, and the amAdmin CLI tool is installed in the <dsame_root>/web-apps/services/WEB-INF/bin directory. The amadmin CLI tool is a shell script wrapper around the actual Java class, which is the actual implementation of this CLI.

Syntax for using the amadmin Tool

The syntax for using the amadmin tool follows:

NOTE You must enter two hyphens (not one) exactly as shown in the command lines instructions in this section.

```
amadmin --runAsDN <dnname> --password <password>
[--locale <localename>][--verbose|--debug]--data
xmlfile1 [xmlfile2 ...]

amadmin --runAsDN <dnname> --password <password> [--locale localename]
[--verbose|--debug]--schema xmlfile1 [xmlfile2 ...]

amadmin --runAsDN <dnname> --password <password> [--locale localename]
[--verbose|--debug]--deleteService <serviceName> [<serviceName2> ...]

amadmin --help

amadmin --version
```

Detailed descriptions of all the above-mentioned arguments is in the next sub-section.

Syntax Description for the amadmin Command Line Interface Tool

Following are syntax descriptions for the amadmin command:

`--runAsDN`

is a mandatory argument and its value is <dnname>. Replace <dnname> with your own DN. <dnname> is the DN of the authorized user who is running the `amadmin`. It is used to authenticate to the LDAP server. For example:

```
--runAsDN uid=amAdmin,ou=People,o=iplanet.com,o=isp
```

or you can insert spaces between the different domain components of the DN, and double quote the entire DN:

```
--runAsDN "uid=amAdmin, ou=People, o=iplanet.com, o=isp"
```

`--password`' is a mandatory argument and its value is <password>. Replace <password> with your own password, which should correspond to the DN (Distinguished Name) name provided.

`--locale` is an optional argument and its value is 'localename'. Replace 'localename' by your own 'localename'. This argument is used for the customization of the message language. If this argument is not provided then default is used. Default locale is 'en_US', which is US English.

`--debug` is an optional argument. When used, this option writes messages to the file `amAdmin` which is created under the `/var/{BASE_DIR}/SUNWam/web-apps/services/debug` directory. These messages are more detailed in technical terms. However, the messages written in the `amAdmin` file are not i18n-compliant.

`--verbose` is an optional argument. When used, this option does not print detailed information; instead, it prints the overall progress of the `amadmin` command on the screen. All the messages output on the command line are i18n-compliant.

`--data` option creates, deletes and reads the various directory objects such as roles, groups, organizations, and containers, which are detailed in the XML input files. The --data option also registers, unregisters, activates, and deactivates a service; and gets the number of activated and registered services, and registered and activated service names for an existing organization. You can specify multiple XML files or just specify one XML filename (such as createRequests.xml or deleteRequests.xml) for this option. You must provide either the --data or --schema option depending on whether you are performing batch updates to the DIT, or loading service schema and configuration data.

`--schema` option loads (imports) service schema and metadata (configuration data), which is detailed in the XML input files. When loading service schema information, you can also specify multiple XML filenames, in addition to a single XML filename (such as sampleMailService.xml). You must provide either the --data' or --schema option depending on whether you are performing batch

updates to the DIT, or loading service schema and configuration data. `<xmlfiles1>` is not an argument. `<xmlfiles1>` is the name of a single XML filename or multiple XML filenames to be loaded or imported into the DSAME (and Directory server).

`--deleteService` argument is for deleting a service—its schema only. This service must have been imported previously by using the amadmin CLI tool's --schema option.

`--serviceName1` is the ServiceName which is under the `<ServiceName=...>` tag of the serivce XML file.

`--help` displays the command syntax for the amadmin command.

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

Registering Services in DSAME

1. First step for any service in its lifecycle is to load its schema into DSAME.
2. At this point, no organization has yet registered for this service. So the second step would be to register the service for the organization so that the organization has access to (or can avail of) that service. For example, in a hosted environment, different orgs may purchase (register) different sets of services.

The service should display in the available Services list in the Admin Console GUI under the User management page.

Registering and Unregistering a Service for an Organization

This section provides some information on registering and unregistering a service for an organization. After importing, or loading, service schema and configuration information into DSAME using the amadmin CLI tool, you must first register the service (using the registerRequests.xml file), then you must activate the service (using an activateRequests.xml). Similarly, when deactivating a service, you must first deactivate the service, then unregister the service, in that order.

You use the amadmin CLI tool's -schema option to import (load) a service. After importing the service, the service is created in the "config" (configuration) area of the DIT for a particular organization.

Unregistering a service

Unregistering a service for a particular organization removes the COS definition of the service. This service must have been imported using the amadmin CLI for that particular organization.

Get Number Of Services

To get all services registered for an organization (all active and deactivated), use the `getNumOfServices.xml` file.

Guidelines for Loading Services into DSAME

This chapter provides an overview of service management in DSAME, and includes information on how to write service XML files per the DTD (sms.dtd). Also refer to the Readme.html file which is located in the following directory:

`<dsame_root>/samples/admin/cli`

for more information on how to use the amadmin CLI tool to load services and perform batch updates to the DIT.

Make Sure you have the Necessary Files before Loading a Service

See Chapter 5, “Understanding DSAME XMLs and DTDs" for information on the sample mail service files: sampleMailService.xml, sampleMailService.properties, and sampleMailService.ldif files.

There is a sample mail service located in the `<dsame_root>/samples/admin/cli/sampleMailService` directory, along with an `.ldif` file and .properties file, which you can use to practice loading a service into DSAME. Following are brief descriptions of what you can do with these sample files:

- sampleMailServiceSchema.ldif—The .ldif file contains the LDAP schema for the sampleMailService, which contains the necessary LDAP schema corresponding to the LDAP object classes and LDAP attribute names. You need to load the .ldif file for the service (such as sampleMailServiceSchema.ldif) into Directory Server using the `ldapmodify` command. If an administrator wanted to create a policy based on the sample policy schema defined in this file, for example, he or she would need to create

the LDAP schema corresponding to the LDAP object classes and LDAP attribute names. For the sampleMailService, a sample .ldif file is provided in the `<dsame_root>/samples/admin/cli/sampleMailService` directory.

- In the sampleMailService.ldif file, the OIDs follow the standard convention of <attributename>-oid. When writing your own .ldif files to load the schema, you should use the read OIDs and get them registered before deploying DSAME. For more information about OIDs, or to request a prefix for your enterprise, send mail to the IANA (Internet Assigned Number Authority) at iana@iana.org or visit the IANA website at: http://www.iana.org/iana/.

- sampleMailService.xml—You need to have a service xml file, which defines the necessary service schema and configuration data for your service per the rules defined in the sms.dtd file (for example, the sampleMailService.xml file already contains the necessary schema and configuration data). The service XML file (for example, sampleMailService.xml) defines attributes and object classes, among them i18Nkeys, also called localization keys, which map one for one to corresponding fields in the .properties files.

  If writing a new service, you will need to update an existing schema and configuration data (in an XML file) with the necessary attributes and object classes. You can use this sampleMailService.xml as is, and load using `amadmin`.

- `sampleMailService.properties`—The sampleMailService.properties provides a sample .properties file, which defines the object class for the service profile, and the index keys that are defined in the service XML file (for example, i18nKey="a1" defines the localization key in the sampleMailService.xml file, which tells Admin Console to display that corresponding field on that service's page in Admin Console (in this case, it displays the field "Mail Status" on the Sample Mail Service Profile page in Admin Console).

  Note on sampleService.properties File

  The sampleMailService.properties file is used to display the sample services, attributes, etc. on the Admin Console when a user displays that service's page in the Admin Console.

  When a user or administrator brings up 'Service Management' in Admin Console, the Admin Console code looks for the sampleMailService.properties and gets the appropriate value for the key given and displays all information for that service on the Admin Console screen in the respective language specified.

The path for this .properties file should be set correctly as described in “Specify pathname for sampleMailService.properties in jvm12.conf File,” on page 202.

Extend the Service Schema by Loading the .ldif File

Before creating any specific policies, you must extend the schema by using the `ldapmodify` command, which takes the .ldif file as input. (Also see the Directory server 5.0 documentation for information on updating schema.)

Go to the install directory of Directory Server:

```
cd <DS_INSTALL_DIR>/shared/bin
```

and run:

Code Example 6-1 ldapmodify Command Example Used to Extend the Schema

```
<dsame_root>/SUNWam/web-apps/services/WEB-INF/bin/ldapmodify -a -h
"<hostname>" -p <DS portnumber> -D "<userid to manage DS>" -w "<password>" -f
"<dsame_root>/SUNWam/samples/admin/cli/sampleMailService/
sampleMailServiceSchema.ldif"
```

By default, the “`<userid to manage DS>`” would be "cn=Directory Manager".

If the schema was created, the result of the previous command would be:

```
Modifying entry cn=schema
```

To make sure that the schema has been created, use the `ldapsearch` command as shown below.

Code Example 6-2 `ldapsearch` Command Example to Ensure that Schema has been Created

```
<dsame_root>/SUNWam/bin/ldapsearch -h "<hostname>" -p <DS portnumber> -b
"cn=schema" -s base -D "<userid to manage DS>" -w "<password>"
"(objectclass=*)" | grep -i "iplanet-am-sample-mail-service"
```

If the schema was created, the result of the previous command would be:

Code Example 6-3 Result of `ldapsearch` Command if Schema was Created

```
objectClasses=( 1.2.3.888.23 NAME 'iplanet-am-sample-mail-service' DESC
'iPlanet dpro SampleMail Service' SUP top
AUXILIARY 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' ) )
```

NOTE The order in which you extend the schema (that is, run `ldapmodify` and `ldapsearch`) and then use the amAdmin CLI tool is not important. You can run the amAdmin CLI tool first, then load the schema.

However, after registering a service using the amadmin CLI tool, you should always run `ldapmodify` so that the objectclass of the service will be added to all the user entries. (See "Sample .ldif file that shows the objectclass of a service added to a user entry," on page 204.) Or you can refer to the Directory server 5.0 documentation sections on extending schema for information.

Restart the Directory Server

Restart the Directory Server, which is located in the `/<DS_INSTALL_DIR>/slapd-<hostname>` directory.

1. Go to the directory where the Directory Server is installed:

   ```
   cd /<DS_INSTALL_DIR>/slapd-<hostname>
   ```

2. Restart the Directory Server with:

   ```
   ./restart-slapd
   ```

Specify pathname for sampleMailService.properties in jvm12.conf File

1. Next, modify the `jvm12.conf` file under the path `<dsame_root>/servers/https-<fully-qualified-hostname>/config`

   and add the following directory pathname:

   `<dsame_root>/samples/admin/cli/sampleMailService`

   to the entry `jvm.classpath=`

   All the .properties files should be located in the <dsame_root>/web-apps/services/`WEB-INF/classes` directory, by default.

   or

Alternatively, you can copy the .properties file into the following directory:

```
<dsame_root>/web-apps/services/WEB-INF/classes
```

If you copy the .properties file to the `classes` directory, then you do not need to restart the Web and Directory servers (which is covered in “Start the Servers (Web and Directory Server),” on page 203).

Start the Servers (Web and Directory Server)

NOTE This step is necessary only if changed the classpath (described in “Specify pathname for sampleMailService.properties in jvm12.conf File,” on page 202.)

Restart all servers.

1. Go to `<dsame_root>/web-apps/services/WEB-INF/bin`
2. Start the DSAME server in a terminal window:

   ```
   ./amserver start
   ```

Import the Service XML File(s)

Next, import the service’s XML file, for example, “sampleMailService.xml”.

If the product has been installed in `<dsame_root>/SUNWam`, and Directory Server has been installed in <iDS_root_dir>, then after the product has been installed, follow the steps below to import the service XML file.

1. `cd <dsame_root>/web-apps/services/WEB-INF/bin`
2. `./amadmin -runAsDN uid=amAdmin,ou=People,o=iplanet.com,o=isp -password passwordvalue` -verbose -schema `<dsame_root>/samples/admin/cli/sampleMailService/sampleMailService.xml"`

   After the `sampleMailService.xml` is loaded successfully, a message similar to the one below displays:

Code Example 6-4 Message display after loaded sampleMailService.xml File

```
Entering ldapAuthenticate method!
No Exceptions! LDAP Authentication success!
Calling SSO method!
Calling SCHEMA MANAGER
XML file to
import:/opt/SUNWam/samples/admin/cli/sampleMailService/sampleMailService.xml

Loading Service Schema XML
/opt/SUNWam/samples/admin/cli/sampleMailService/sampleMailService.xml
Calling the constructor ServiceManager!
Reading schema file
:/opt/SUNWam/samples/admin/cli/sampleMailService/sampleMailService.xml
Done loading Service Schema XML:
/opt/SUNWam/samples/admin/cli/sampleMailService/sampleMailService.xml
Successfully completed.
```

Register the Service

Next you must register the service. The preferred way to register a service is to go DSAME console and register the service. Or you could use a command similar to the following to register the sampleMailService (using the sampleMailService.xml).

Code Example 6-5 Command to register sampleMailService

```
<dsame_root>/SUNWam/web-apps/services/WEB-INF/bin/amadmin
-runasdn uid=amadmin,ou=people,o=iplanet.com,o=isp
-password netscape
-debug
-data <dsame_root>/SUNWam/samples/admin/cli/bulk-ops/registerRequests.xml
```

Sample .ldif file that shows the objectclass of a service added to a user entry

You only need to perform this step if users already exist.

If you create users after the service has been loaded, you can skip this step. (This step provides information on what to do if users already exist, and you must add the objectclass to the users. If users do not exist, and you create after you have loaded the service, then this is done automatically for you, and you do not have to add the objectclasses.

If you have to perform batch updates to user objects in the DIT, you can use the migration scripts provided in DSAME 5.0. These migration scripts are described in the *iPlanet Directory Server Access Management Edition Installation and Configuration Guide*.

Below shows some sample ldif that shows the objectclass of the service "iplanet-am-sample-mail-service" added to a user entry, specifically "shiva", after the schema has been extended. For example, you might create an `.ldif` file named `xyz.ldif` with the entries below:

Code Example 6-6 Sample .ldif code that shows objectclass of `iplanet-am-sample-mail-service` service added to user entry

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

Then run the `ldapmodify` command as shown here:

Code Example 6-7 ldapmodify Command Example

```
<dsame_install_dir>/SUNWam/web-apps/services/WEB-INF/bin/ldapmodify -a -h
"<hostname>" -p <Directory Server  portnumber> -D "<userid to manage DS>" -w
"<password>" -f " <path of the xyz.ldif file>/xyz.ldif"
```

Add the sampleMailService to the Service Hierarchy

The Admin Console displays only DSAME-related services. If you are adding new (customized) services, then you must add them to the service hierarchy so you can view them on the Admin Console's Service Management page.

NOTE The SampleMailService samples provided are for instructional purposes only to learn how to create a service and load it into DSAME (and update the DireTctory server with any new / modified objectclasses). In the DSAME 5.0 release, you cannot integrate your iPlanet Messenger service. This will be supported, however, in a future release.

To display the sampleMailService in the Admin Console, you need to add the sampleMailService to the service hierarchy so that it will display under *Other Configuration* on the Service Management page. To do this, do the following steps:

1. Bring up the DSAME Admin Console by entering the URL into a web browser, for example,

   `http://sparkie10.red.iplanet.com:8080/console`

   and logging into DSAME Admin Console as `amadmin` (as the top level administrator or Super-administrator).

2. In Admin Console, select *Service Management*.
3. Select the arrow for *Administration* under *DSAME Configuration*.
4. Scroll down to *Service Hierarchy* which is under *Administration - Global*.
5. Enter

   `other.configuration|sampleMailService`

   into the textfield and click *Add*.

6. Click *Submit*.
7. Log out of Admin Console.
8. Log into Admin Console again.
9. Go to the Service Management page, and view the sampleMailService and its profile on the ServiceManagement page.

After you have done all the above procedures (that is, modify the jvm12.conf file, import the service, then register and activate the service), you should be able to create templates through the Admin Console, and then add new roles and users, and assign policy attributes to them.

Administration Service Attribute (iplanet-am-admin-console-service-hierarchy)

This section explains what the `iplanet-am-admin-console-service-hierarchy` attribute is used for. The part about DSAME services would be for DSAME developers or someone like an ISV who would adding an external or custom service that would be considered part of DSAME. A good example of this would be an authentication service. An authentication service is actually an addition to the services of DSAME, even though it is not shipped with DSAME. (This might also be referred to as an "external service", which means that it is not shipped as an internal service with DSAME, such as Logging, Session, and Naming services.)

If the new service is something that is not an internal DSAME service (that is, shipped with DSAME), then it would go under `Other services`. An example of this would be something like a Calendar application or Mail service.

The first method could be used if someone loaded an XML service using `amadmin` and then needs to add it to the hierarchy. They would use the Admin Console to add their service to the hierarchy.

After making modifications to the `amAdminConsole.xml`, run

```
amadmin -deleteService amAdminConsole

amadmin -schema amAdminConsole.xml
```

This is also an option as with any service. Customization engineers or administrators can make their modifications through the Admin Console or command line. Note that this is only for the default values. Other XML components like `type, syntax,` and `i18n` keys must be modified using the `amadmin` CLI tool. Typically, customization engineers and administrators will only do this during installation by using the `amadmin` CLI tool.

There are two ways to have a new service shown in a hierarchical manner.

First (preferred) method

Go to the DSAME Admin Console, click on the Service Management view, click on Administration Service and look for the Service Hierarchy list. Update the list according to the format described in the section “Second Method (Alternative),” on page 208.

The second method allows you to import the service hierarchical information to Directory Server upon installation. The first method allows you to change the hierarchical information after you have imported, or loaded, the service into DSAME.

Second Method (Alternative)

This method is only applicable if you are installing DSAME for the first time. Only use this method if you have not changed the defaults for the Admin Console service (`amAdminConsole.xml`), or if you have reverted back to the installation defaults.

Only use the first method if you know that you have not changed the `amAdminConsole.xml` service. The reason for this is because at installation time, defaults are loaded. Then the administrator can change some parameters for the service, which modifies the schema in the XML in the Directory server. Then if you were to reload this service, you would overwrite all those changes that had been made previously to the XML in the Directory server.

After adding an entry to the:

`com/iplanet/dpro/admin/xml/amAdminConsole.xml`

under the Configuration tag `iplanet-am-admin-console-service-hierarchy` attribute, if it is a DSAME service, it will be:

`dsame.configuration|<serviceName>`

<Value>dsame.configuration|iPlanetAMAdminConsoleService</Value>

If it is a DSAME authentication service, it will be

```
dsame.configuration|authentication|<serviceName>
```

```
<Value>dsame.configuration|authentication|iPlanetAMAuthAnonymous
Service</Value>
```

If it is a third party service (versus being an "internal DSAME server", such as Naming service, Session service, or Logging service), it will be:

```
other.configuration|<serviceName>
```

For example:

```
<Value>other.configuration|XYZService</Value>
```

Assign Policies to the Sample Mail Service

The following steps give some information on how you would assign policies to a service after loading it into DSAME Admin Console.

1. Log in to the Admin Console as `amadmin`, if you are not already logged into Directory Server Admin Console.
2. Go to the Policy Management page for o=iplanet.com.
3. Press the Create button.
4. On the Properties page for the Sample Mail Service Profile, select the value of the priority to higher, and the mail quota to 8888.
5. Create the service in the o=iplanet.com organization.
6. Go to the User Management page for o=iplanet.com.
7. Create a new role as “role1”.
8. Create a new user as “user1”.
9. Assign user “user1” to the role of “role1”.
10. Select roles, select role1, then select Services.
11. Click on Properties for the sampleMailService to create a template.
12. Go to the Policy Management page.
13. Go to the organization `o=iplanet.com`.
14. Select role “role1”.
15. Select services.
16. Click on the “sampleMailService” properties.
17. Create a policy template, and change the value of the priority to "highest" and the mail quota to 6666.
18. Do an `ldapsearch` for “user1”. To do this, go to the directory `/<DS_INSTALL_DIR>/shared/bin/` and run:

    ```
    ./ldapsearch -h localhost -p 389 -b
    "uid=user1,ou=People,o=iplanet.com,o=isp -D "cn=Directory
    Manager" -w password "(objectclass=*)"
    ```

19. Check that the attributes of the “sampleMailService” are present and that the mail quota = 6666.

View the Policy Profile for a Service that has been added to DSAME

This section gives a procedure on how to view the policy profile of an added service for an organization on the Admin Console's Policy Management page.

If you did not create Policy through the amadmin CLI tool, do the following:

After importing the service using `amadmin`, then creating user management objects amadmin, then registering and activating the service through amadmin:

1. Log into the DSAME Admin Console by entering the URL into a web browser, for example,

   ```
   http://sparkie10.red.iplanet.com:8080/console
   ```

   and logging into DSAME Admin Console as `amadmin` (as the top level administrator or Super-administrator).

2. Click Policy Management.
3. Select the organization.
4. Click Policies.
5. Choose the service from the choice box.
6. Type in the policy name and then select Create.

   The policy template displays.

7. Make your changes to the policy template and click Submit, if needed.
8. Log out of the Admin Console.

   Log in and then repeat Step 2 through Step 4.

9. Choose the policy you have created and view its profile.

If you created Policy through the amadmin CLI tool, do the following:

After importing the service using `amadmin`, then creating user management objects amadmin, then registering and activating the service through amadmin:

1. Log into the DSAME Admin Console by entering the URL into a web browser, for example,

   ```
   http://sparkie10.red.iplanet.com:8080/console
   ```

   and logging into DSAME Admin Console as `amadmin` (as the top level administrator or Super-administrator).

2. Click Policy Management.
3. Select the organization.
4. Click Policies.
5. Choose the policy you have created and view its profile.

View the Profile for an Added Service

This section gives information on how to view the profile for an added service for an organization in the Admin Console's User Management page

After importing, registering, and activating the service, log into the Admin Console.

1. Select *User Management*.
2. Select the organization for which the service is registered.
3. Select *Services.*
4. Click the service for which you want to view the profile.

   The message 'There is no template for this service' displays.
5. Press Create to create the template.
6. Log out of Admin Console.
7. Log in again and repeat steps Step 1 through Step 4.

   The dynamic attributes and the profile for the added service for that organization displays.

Guidelines on Performing Batch Updates to User Objects in Directory Server

This section provides an overview how you typically would perform batch updates on user objects (such as groups, users, roles, people containers, etc.) in the Directory Server. You can use the sample XML files provided in the following directory:

```
<dsame_install_dir>/SUNWam/samples/admin/cli/bulk-ops/CreateRequests.xml
                                                     GetRequests.xml
                                                     Requests.xml
```

```
deleteOrgRequests.xml
deletePCRequests.xml
deleteGroupRequests.xml
registerRequests.xml
unregisterRequests.xml
activateRequests.xml
deactivateRequests.xml
getActivatedServices.xml
getRegisteredServices.xml
getNumOfServices.xml
```

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

List of Sample XML Files for Performing Batch Updates to DIT

Sample XML input files for -data option are provided in the following directory:

`<dsame_root>/SUNWam/samples/admin/cli/bulk-ops/`

The xml files and brief descriptions of their purpose follows:

createRequests.xml—Creates all the objects in the Directory Server.

getRequests.xml—Gets information about all objects (previously created by createRequests.xml).

deleteOrgRequests.xml—Deletes all objects (previously created by createRequests.xml) under the Organization.

deletePCRequests.xml—Deletes all objects (previously created by createRequests.xml) under the People Container.

deleteGroupRequests.xml—To delete all objects (created by createRequests.xml) under Group.

registerRequests.xml—Registers a service for an existing organization. This service must have been imported using the amadmin tool.

unregisterRequests.xml—Unregisters a service for an existing organization. This service must have been imported using the amadmin tool. (If a service has been registered, then activated, it must first be deactivated, then unregistered.)

activateRequests.xml—Activates a service for an existing organization. This service must have been previously imported and registered by using the amadmin tool.

deactivateRequests.xml—Deactivates a service for an existing organization. This service must have been imported and registered using the amadmin tool.

getActivatedServices.xml—Gets the list of activated service names for an organization.

getRegisteredServices.xml—Gets the list of registered service names for an organization.

getNumOfServices.xml—Gets the total number of registered and activated services for an existing organization.

NOTE All of these xml files perform operations on the dit, in that they create, delete, or get attribute information on user objects, such as organizations, roles, groups, people containers, or users.

Steps to Perform Batch Updates to DIT

Following is a list of high level steps and things to do when performing batch operations on Directory Server through DSAME's amadmin tool.

Define user objects in createRequests.xml File

1. First define the user objects you want to create in Directory Server in the createRequests.xml file, provided in the following directory:

   `<dsame_root>/samples/admin/cli/bulk-ops`

   The createRequests.xml file creates groups in the default mode (non-compliant DIT and schema).

Changes to make if the DSAME product is installed in Compliant mode (iPlanet DIT and schema mode)

There are now two modes for the product:

1. iPlanet DIT and schema mode (compliant)

   If you have installed in the “Compliant mode” (that is, iPlanet DIT and schema mode) you must modify the createRequests.xml file to create the groups under ou=Groups,o=sun.com. (Replace the actual organization name for sun.com, shown in this example.)

```
<CreateGroups>
        <DN>ContractorsGroup,ou=Groups,o=sun.com</DN>
        <DN>EmployeesGroup,ou=Groups,o=sun.com</DN>
</CreateGroups>
```

   iPlanet DIT and schema (Compliant) mode dictates that o=ISP be a child of the root suffix, which must be entered during the installation process. All orgs will then be created under o=ISP,<root suffix>.

2. Default mode (non-compliant)

   In default mode ,the 'ou=Groups' is not created. 'ou=Groups' is created only when running in the iPlanet DIT and schema (Compliant) mode. The createRequests.xml file in the `samples/admin/cli/bulk-ops` directory shows the sample XML format for creating groups in the default mode (that is, non-compliant mode).

   Therefore, you must modify all related 'o=isp' in the sample XML files as 'o=ISP,o=xyz', with 'xyz' being the root suffix entered by the administrator during installation of DSAME in iPlanet DIT and schema (compliant) mode.

Load the Batch Update Defined in the XML File into DSAME

To load an XML file that defines some batch operations such as creating, deleting, or getting (reading) user objects (such as organizations, people containers, roles, users, groups) in the DIT, run the amadmin command with the -data option.

1. Go to the following directory:

   `<dsame_root>/SUNWam/web-apps/services/WEB-INF/bin`

   This directory contains the `amadmin` executable.

2. Run the following command:

Code Example 6-8 amadmin Command to load Batch Update to DIT file (createRequests.xml)

```
./amadmin -runAsDN uid=amadmin,ou=People,o=iplanet.com,o=isp -password
<password> -verbose -data
<dsame_root>/SUNWam/samples/admin/cli/bulk-ops/createRequests.xml
```

Verifying that the DIT has been Populated Correctly

If you want to verify that the DIT has been populated correctly in Directory Server , do the following steps:

1. cd /<DS_INSTALL_DIR>/slapd-<hostname>
2. Export the Directory Server contents into an .ldif file.

   `db2ldif -s o=isp`

   This would result in displaying the name of the ldif file stored under

   `/<DS_INSTALL_DIR>/slapd-<hostname>/ldif`

   View that file to ensure that all the objects (described in the `createRequests.xml` file) with their attributes and values were created.

Verification Caution

Typically, when you use `amadmin` to perform batch operations on Directory Server. Even without the additional verification, such operations may sometimes takes hours, and in extreme cases, even days.

So you should be careful that the additional verification will only add additional hours or even days, depending on the amount of data.

Also, when you perform a verification, you should start the dump on the appropriate subtree instead of on the entire tree, which would take longer, and be unnecessary.

When performing a verification action, be aware that When amAdmin CLI tool is normally used, it is for batch operations. Even without the additional verification, such operations may sometimes takes hours and in extreme cases, even days.

Be careful that the additional verification operation will only add additional hours or even days, depending on the amount of data. Also, when you verify, you should start the dump on the appropriate subtree rather than the entire tree.

View the .ldif File to Ensure that the objects were created in the Directory server

View the .ldif file in <DS_INSTALL_DIR>/slapd-<hostname>/ldif directory to ensure that all the objects (described in createRequests.xml file) with their attributes and values were created. You can do this with any text editor or viewer.

Tips when running amadmin Tool

Following is some useful general information that involves using the amadmin CLI tool.

NOTE You cannot set ACIs through the amAdmin CLI tool.

Using ldapmodify versus the DSAME amadmin Tool

Both `ldapmodify` and the amadmin CLI tool "eventually" end up using LDAP SDK/JDK to communicate with Directory server. So, in theory, whatever you can do with the amadmin CLI tool can be done with `ldapmodify` (not the other way round). Primarily, the benefits of CLI are ease-of-use, higher-level abstraction and XML-compliance. Other than creating users, it is not recommended that you use `ldapmodify` to create entries for DSAME.

Benefits of using CLI and XML Files

When creating certain higher-level DSAME abstractions (such as policies, service management), you do not have to learn or know how these abstractions are "implemented" or "mapped" to the DIT. Not all DSAME abstrations map 1-1 with Directory Server entries. For example, there may be more than one Directory Server entry manipulated as a result of some higher-level user operation. So users also don't have to know the "inter-dependencies" between different entries.

The CLI tool will ease such tasks by avoiding any problems with inconsistent updates of Directory Server using ldapmodify. With ldapmodify, customers may have to know and list all the LDAP object classes and relevant LDAP attribute names. DSAME CLI hides most of the low-level details of LDAP by allowing the customer focus on tasks such as creating users, adding users to roles / groups, etc.

Most customers (developers) are more familiar with XML syntax than with the ldif syntax. For such customers who are comfortable with XML, they can continue to use XML instead of learning a new / one-more syntax. The availability of XML editors "sometimes" helps customers write XML files and validate the syntax right away.

How to Determine Attribute/Value Pairs to Provide in the XML Files

When creating user objects in the Directory, such as organizations and people containers, it may be difficult to determine what attribute / value pairs to give in the XML files. You can set or specify any attributes that are specified in the DSAME configuration file's CreationTemplates. (Currently, the DSAME configuration file is called ums.xml or umsCompliant.xml, and is located in `<dsame_root>/SUNWam/config/ums.xml`.) Every entry that DSAME creates must have a corresponding CreationTemplate, which instructs the User Management component of how to create that entry. It specifies what objectclasses and attributes are mandatory and optional and what default values should be set, if any.

If a deployment engineer or customization developer wanted to add customized object classes to DSAME, they would need to modify the templates in the DSAME configuration file (ums.xml or umsCompliant.xml) so that both DSAME and Directory Server can read and recognize the new object classes and attributes. Then, to manage them from the DSAME Admin Console, these new object classes and attributes have to be defined and "modelled" in the XML format, then imported into DSAME using the procedures described in this chapter.

Which XML Files are Used for DSAME User Management

Any attribute/value pairs that Directory Server must know about, that is, read, store, or perform some operation on, must be defined in the ums.xml file. Any attribute/value pair that is to be displayed in DSAME Admin Console's User page must also be defined in the dpUser.xml file. The attribute/value pairs defined in the ums.xml file defines all the possible user objects that Directory server can create, read, and search on. For the attribute to be displayed in Admin Console, you must define those in the "user" subschema in the dpUser.xml file. Also, a corresponding .properties file must be created, with accurate i18nKey fields (called "localization keys" or "index keys") which point to actual field names to be displayed on that User server page in Admin Console.

Explanation on Defining GetUsers in amAdmin.dtd

This section provides information when performing batch updates to the DIT.

To give an example and description of what DNsOnly=true|false means, for those objects that may have LDAP attributes, all 'get' operations follow the same design pattern which is given below:

If the element has an XML attribute 'DNsOnly' set to true or doesn't have that XML attribute, only the DNs of the corresponding DSAME objects shall be returned.

If DNsOnly="false", the entire object (with the LDAP attribute value pairs) shall be returned.

However, the behavior of DNsOnly is valid ONLY if there are no child elements (DNs) specified.

If the DNs are specified, the entire object shall always be returned.

Additionally, say if an administrator or user wanted to list two users, syntax for 'GetUsers' in amAdmin dtd is the following:

Code Example 6-9 GetUsers Element in amAdmin.dtd

```
<!ELEMENT GetUsers (DN)* >
<!ATTLIST GetUsers
    DNsOnly    (true | false) "true"
>
```

Code Example 6-9 GetUsers Element in amAdmin.dtd

```
You should mention or list the DN's between the opening and
closing tag for GetUsers. Thus you would specify in the XML file
something like the following:
<OrganizationRequests DN="o=isp">
         <GetUsers DNsOnly="true">
              <DN>uid=amAdmin,ou=People,o=iplanet.com,o=isp</DN>
               <DN>uid=dpUser,ou=People,o=iplanet.com,o=isp</DN>
         </GetUsers>
</OrganizationRequests>
```

All Files Input with the amadmin Tool must be XML Files

You cannot use the amadmin CLI tool with anything other than XML input files. All requests (commands) to the amadmin tool are imported or fed into DSAME through XML input files. Thus the amadmin CLI tool is not involved in any operations that do NOT involve XML files. If an administrator wants to populate the DIT in Directory Server with user objects (create roles, groups, people containers, etc.), or perform batch reads (gets) or deletes on the Directory server DIT, then he or she must be able to write the necessary XML input files based on the amadmin data DTD or SMS schema DTD.

Using amadmin vs. DSAME's Admin Console

The primary purpose of the amadmin CLI tool is to enable administrators to perform batch operations possible for the case where users want to import a significant amount of data into the database. This is very time-consuming if Admin Console is used. You would have to perform each creation, deletion, or read operation one at a time.

The amadmin tool only supports a subset of features that Admin Console supports.

You can only perform service registration (that includes importing service schema and configuration data through XML input files) through the amadmin tool.

CAUTION This process may take hours if the number of users is large.

Service Registration XML DTD

For information on the service registration XML DTD, see Chapter 5, “Understanding DSAME XMLs and DTDs". It provides some information on the `sms.dtd` elements and attributes that DSAME uses, and describes how you would use the supported attributes types to create a custom service.

When importing or loading any new or customized services to DSAME, you must write a service XML containing service schema and configuration data that corresponds to this DTD.

This DTD is located in the following directory:

```
<dsame_install_dir>/SUNWam/web-apps/services/dtd
```

NOTE When performing service registration through `amadmin`, the `sms.dtd` is used, along with the service XML files, for example, `sampleMailService.xml`. When performing batch updates to the DIT, the `amAdmin.dtd` is used, along with the batch update XML files, such as `createRequests.xml` file.

Deleting a Service that has been Registered and Configured

If a service (having dynamic attributes) has been registered and configured in Admin Console for several organizations and roles, and it is in use, an administrator can delete this service. It can be deleted through Admin Console or by using the `--deleteService` option of amadmin tool.

You should not delete the DAI Service (ums.xml configuration file)

To modify the ums.xml configuration file after installation of DSAME, use the Directory server Console.

DSAME does not support deleting the DAI service (ums.xml configuration file). The way to modify the DAI service (ums.xml) is either through LDAP (modifying an .ldif file directly) ro through the Directory Server Console. The reason that deleting and reloading this file through the amadmin CLI tool is not supported is that if the smallest error introduced into this file when making modifications may cause the DSAME platform to not initialize.

You may, however, delete the ums.xml configuration through the Directory Server Console (it’s called DAI service) and then reimport the file using amadmin.

It is recommended that any modifications made to the ums.xml configuration file be done through the Directory Server Console, as this method is less error-prone than modifying the ums.xml file directly.

Basedn for DAI Service Tree

Following is the location and basedn for the DAI service tree in Directory Server:

```
ou=DAI,ou=services,<install root suffix>
```

Location in Directory Server for User creation template for adding service object classes

Following is the location in Directory Server for the User creation template (for when adding service object classes):

```
ou=BasicUser,ou=CreationTemplates,ou=templates,ou=1.0,ou=DAI,
ou=services,<install root suffix>
```

Location in Directory Server for Organization creation template for different org object classes and naming attributes

Following is the location in Directory Server for the Organization creation template (for when adding different organization objectclasses and naming attributes):

```
ou=BasicOrganization,ou=CreationTemplates,ou=templates,ou=1.0,
ou=DAI,ou=services,<install root suffix>
```