www.novell.com/documentation

Novell Modular Authentication Services

Developer Kit

March 19, 2012

Novell.

Novell, Inc.
1800 South Novell Place
Provo, UT 84606
U.S.A.
www.novell.com

Online Documentation: To access the online documentation for this and other Novell developer products, and to get updates, see developer.novell.com/ndk. To access online documentation for Novell products, see www.novell.com/documentation.

Contents

About This Guide

Novell® Modular Authentication Services (NMAS™) is a component of Novell eDirectory™ that enables you to centrally manage multiple authentication methods across your network. The NMAS SDK provides a set of tools to create an expanded set of NMAS login methods to help you secure critical network resources.

This guide contains the following sections:

Audience

This guide is intended for advanced application developers who are familiar with NMAS, other Novell technologies, and Java, C and C++ development environments.

Feedback

We want to hear your comments and suggestions about this manual. Please use the User Comments feature at the bottom of each page of the online documentation and enter your comments there.

Documentation Updates

For the most recent version of the *NMAS NDK Guide*, visit the NMAS NDK Web site (http://developer.novell.com/wiki/index.php/Novell_Modular_Authentication_Service).

Additional Documentation

For the most recent version of the NMAS Installation, Administration, Login Method, and Novell RADIUS Administration guides, see Novell Modular Authentication Services (NMAS) (http://www.novell.com/documentation/nmas21/index.html).

Documentation Conventions

In Novell documentation, a greater-than symbol (>) is used to separate actions within a step and items in a cross-reference path.

A trademark symbol (®, ™, etc.) denotes a Novell trademark. An asterisk (*) denotes a third-party trademark.

When a single pathname can be written with a backslash for some platforms or a forward slash for other platforms, the pathname is presented with a backslash. Users of platforms that require a forward slash, such as Linux or UNIX, should use forward slashes as required by your software.

1 Getting Started

The Novell® Modular Authentication Services (NMAS™) SDK is a development framework that enables your applications to authenticate to Novell eDirectory™ using various login methods. Some of the login methods you might develop include face recognition, fingerprints, voice recognition, signature, iris recognition, certificates, tokens, smart cards, and passwords.

This section explains how to get started with NMAS and contains the following topics:

- Section 1.1, "Development Overview," on page 9
- Section 1.2, "Selecting a Compiler," on page 11

1.1 Development Overview

The NMAS SDK provides the tools to design a flexible and expandable login system using modular plug-in methods that leverage existing Novell security components that are integrated with eDirectory. These integrated security services include cryptography (Novell International Cryptographic Infrastructure (NICI)), secure credential storage (Novell SecretStore®), and authentication (NMAS).

NMAS dynamically assigns login rights to users, groups, and applications according to an established Novell graded authentication (GA) policy.

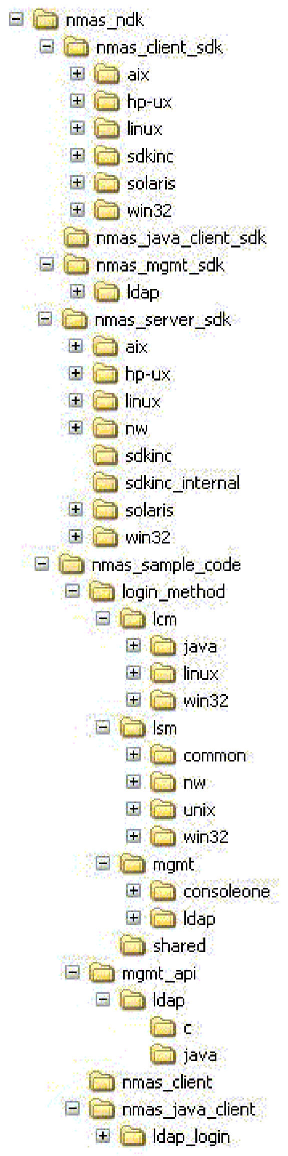
An NMAS operation, in conjunction with graded authentication, allows network administrators to control access to information based on how users log into a system. NMAS stores secret authentication information-passwords, biometric vectors, smart card access, etc.- in the NMAS Authentication Store, a high security data store in eDirectory that only NMAS can read and decipher.

IMPORTANT: To manage and coordinate the development of your login methods to the NMAS framework, you must register with Novell Developer Support. For more details about coordinating the development of your NMAS login methods, please contact Kamal Narayan, NMAS Product Manager (mailto:nkamal@novell.com).

The NMAS executables (that is, NetWare nmas.nlm (release) and nmasdbg.nlm (debug)) are included in the Novell Developer Kit (NDK). However, the complete NMAS package, including the install, is not included in the NDK.

The following diagram illustrates the directory structure of the NMAS SDK:

Figure 1-1 *NMAS SDK Directory Structure*



```
nmas_ndk
    nmas_client_sdk
        aix
        hp-ux
        linux
        sdkinc
        solaris
        win32
    nmas_java_client_sdk
    nmas_mgmt_sdk
        ldap
    nmas_server_sdk
        aix
        hp-ux
        linux
        nw
        sdkinc
        sdkinc_internal
        solaris
        win32
    nmas_sample_code
        login_method
            lcm
                java
                linux
                win32
            lsm
                common
                nw
                unix
                win32
            mgmt
                consoleone
                ldap
            shared
        mgmt_api
            ldap
                c
                java
        nmas_client
        nmas_java_client
            ldap_login
```

Before installing and using the NMAS NDK package, developers should install the NMAS product, then drop in the NDK debug executables for debugging purposes. For current, complete NMAS information, see the NMAS Product Page (http://www.novell.com/products/nmas).

1.1.1 Additional Information

For more information about NMAS and other Novell security products, see the following links:

- Developer Notes article announcing NMAS (http://support.novell.com/techcenter/articles/dnd20000303.html)
- Novell Security, Identity, and Access Management products (http://www.novell.com/solutions/securityandidentity/).

1.2 Selecting a Compiler

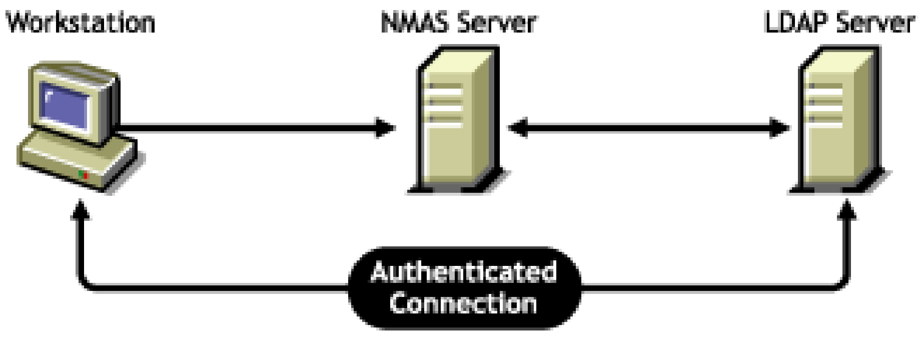
NLM™ applications can be developed on the following compilers.

Compiler	Description
Eclipse	An open source Java*-based IDE designed as a universal development toolset that integrates well with Novell Linux Desktop (NLD). For more information, see Eclipse (http://www.eclipse.org/).
GNU C/C++	A compiler used on many UNIX* and Linux* systems, which also supports NLM development. For more information, see GNU Compiler Collection (http://gcc.gnu.org). For information about a preconfigured environment for NetWare, see the following Forge project: UNIX Environment and gcc for NetWare (http://forge.novell.com/modules/xfmod/project/?aunixnw).
Metrowerks* CodeWarrior*	A commercial compiler with a source level debugger. Certain levels of the DeveloperNet® program include a copy of CodeWarrior or allow you to purchase one at a reduced price. For details, see the Novell DeveloperNet Program (http://developer.novell.com/brochure). For instructions on setting up CodeWarrior for NLM development, see *Targeting the NetWare Operating System PDK 5.0* (http://developer.novell.com/wiki/index.php/Metrowerks_CodeWarrior) from Metrowerks.
Watcom*	A free compiler and the first compiler to support NLM development. It supports developing C applications for NetWare, but does not support C++ applications for NetWare. WLink does not support symbol prefixing except through the use of the ALIAS link directive. Because "@" is overloaded in linker syntax, the solution is difficult and requires quoting. Do not attempt to include Watcom libraries. For more information, see OpenWatcom (http://www.openwatcom.org).

2 NMAS Concepts

Novell® Modular Authentication Services (NMAS™) provides a means of integrating multiple login services with all systems that rely upon Novell eDirectory™, the Novell directory services, as shown in the following overview:

Figure 2-1 *NMAS Login Components*



The NMAS server enables a user to supply identity using a variety of login methods, which are then used to provide an authenticated connection to eDirectory. The NMAS NDK enables you to create your own NMAS login methods. See Figure 2-2 on page 15 for a more detailed view of the architecture involved in this process.

The concepts required for you to create your own NMAS login methods in this SDK are discussed in the following sections:

- Section 2.1, "NMAS Login Considerations," on page 13
- Section 2.2, "Login Method Overview," on page 14
- Section 2.3, "NMAS Client," on page 16
- Section 2.4, "NMAS Server," on page 18

2.1 NMAS Login Considerations

NMAS works with the graded authentication (GA) capability first shipped with NetWare® 5, and provides a common point of administration for all login methods and policies through eDirectory. An NMAS operation, in conjunction with graded authentication, allows the administrator to control access to information based on how a user has logged into a system.

GA provides the ability to associate varying "clearances" to connections on the basis of network policy, such as the login protocols and methods used, the properties of the workstation, the requested capabilities, and other defined policies.

Authentication clearances can be based on any of the following factors:

- Something you know (a password or personal identification number [PIN])

- Something you hold (a token, smart card, or certificate)
- Something you are (a biometric characteristic; for example, voice, fingerprint, retinal scan, or facial recognition.)

Login strength can be improved by combining one or more of these factors. A problem for the network administrator has been how to combine these factors from various vendors into a single composite login for the administrator's network. NMAS was designed to solve this problem.

Through a combination of login and post-login methods, an administrator can define a login sequence that a user must follow in order to log in to the network. Multiple factors can be combined from various vendors. The administrator also can define multiple login sequences through an NDS® login policy. This policy allows access to resources on the network to be controlled based on which sequence is used through Novell Graded Authentication Services (GA).

NMAS supports a combination of login methods including those that allow for NDS password, clear text password, biometric devices, X.509 certificates (digitally signed), message digest, smart cards, and tokens.

It is expected that many login methods will be developed in a partnership with third-party developers. An administrator implementing NMAS can select various login devices from third-party developers and use them together to implement a login sequence for the network.

For more information about installation and administration of NMAS, see the NMAS product portal (http://www.novell.com/documentation/nmas21/).

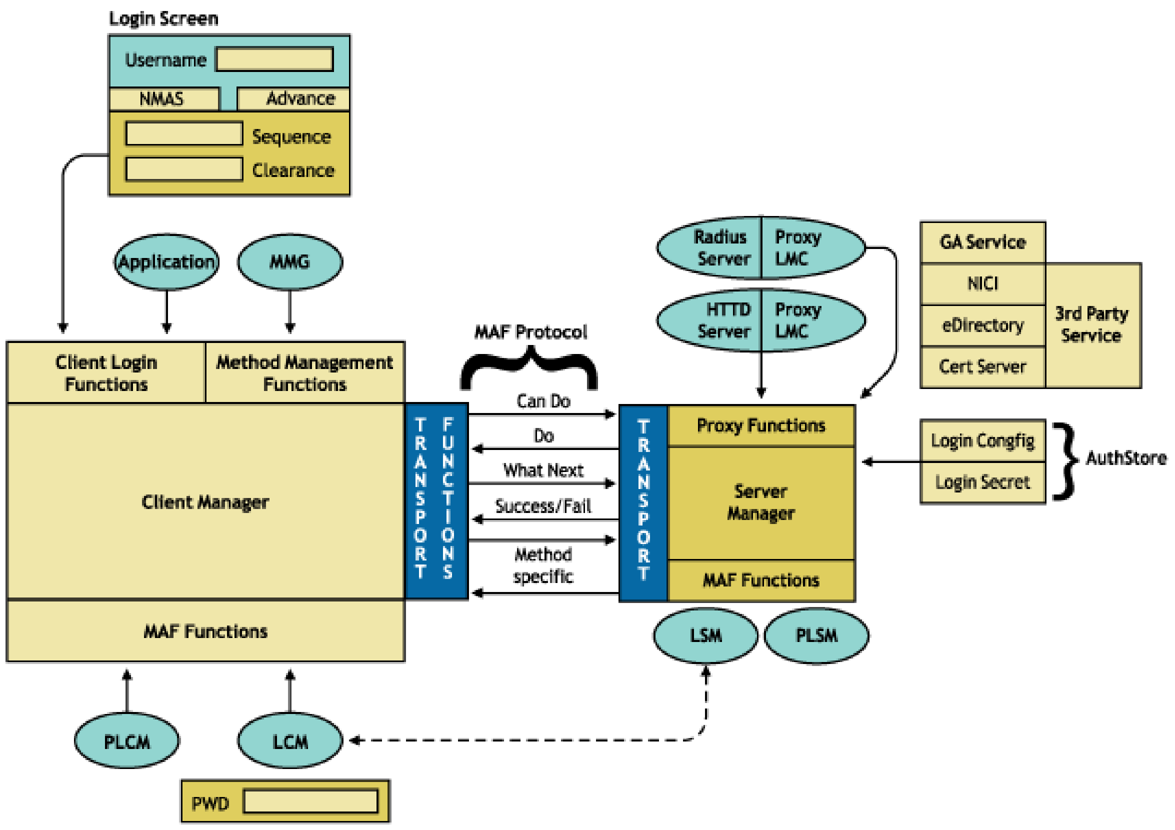
2.2 Login Method Overview

NMAS is designed as a flexible and expandable login framework that enables you to develop plug-in modules, called login methods, for your own login system. These login methods are used to authenticate a user to eDirectory. A complete login method includes the following components:

- **Login Server Module (LSM):** The server-side component of a login method. The LSM is invoked by the NMAS Server Manager. The LSM works with the Login Client Module to transmit login credentials using the multiple authentication framework functions in a client/server model. For more information, see Section 3.1, "Building an NMAS Login Method," on page 23.
- **Login Client Module (LCM):** A client-side component of a login method. The LCM is essentially a program running on the workstation that interacts with the LSM. The LCM and LSM transmit login credentials using the multiple authentication framework functions. See Section 3.5, "Building an NMAS LCM," on page 34.
- **Method Management Tool (MMT):** Most login methods require a management tool to enable system administrators to edit system level login method parameters, which might include user-specific data such as passwords, biometric data, and certificates. These management tools can be implemented as any or all of the following: Novell iManager plug-ins, Novell ConsoleOne snapins, or standalone utilities. In order to provide a consistent user experience, Novell provides iManager™ plug-ins and/or ConsoleOne™ snap-ins for each Novell login method. Many Novell partners also follow this pattern. A ConsoleOne example is included in the sample method directory. For iManager plug-in development, we refer you to the NDK: iManager 2.7 Developer Kit.
- **Login Method Storage Attributes:** Novell defines a set of attributes for storing login method data. Java APIs for storing and reading data on these attributes are provided in the NmasToolkit.jar. These APIs require an SSL LDAP connection with rights to the object on which the data is stored. These APIs encrypt the data upon storage and decrypt it upon retrieval.

All of these components reside on the NMAS Client or NMAS Server, as shown in the following diagram:

Figure 2-2 *The NMAS SDK Architecture*



The login screen prompts the user for identity and optionally requested login sequence and requested clearance. The transport component provides the communication channel between client and server (for example, Winsock, LDAP, and NCP™). The NMAS Client provides the client side of the MAF Protocol (which negotiates the login sequence to be used), invokes the LCMs as directed by the NMAS Server, and provides an API set that can be used by the LCMs to communicate with the LSMs.

IMPORTANT: A login method developer must provide an LCM and an LSM. Currently, the NMAS Client runs on Windows* and Linux platforms. An implementation of the NMAS Client ties into the Novell Client32™, although the NMAS Client does not require Novell Client32.

NMAS stores secret login information—such as passwords, biometric vectors, and other credentials—in the authentication store, a high security data store that only it can read and decipher. Additionally, some NMAS methods rely upon PKI_Store for storage of X.509 certificates.

2.2.1 NMAS Login Method Security Considerations

Consider the following security information as you develop your NMAS Login Methods.

- **Prevent Spoofing of Novell eDirectory®:** It is highly recommended that the Login Server Module (LSM) validates the user-supplied login data against the user login data stored in eDirectory. This prevents unauthorized users from using a rogue Login Client Module to spoof and log into eDirectory. The Login Client Module (LCM) should collect login data from the user and provide the login information to the LSM for validation.
- **Protect user secrets stored in eDirectory:** Protect user login data stored in eDirectory from being read or modified by unauthorized users, including system administrators. Use the NMAS Login Configuration Store or the NMAS Login Secret Store to store encrypted login data.

  The Login Secret Store is intended to store login data, which can be stored by the administrator or the user but not read by the administrator or the user. The Login Configuration Store is intended to store login data which can be stored and read by the administrator or the user.
- **Report login operations accurately:** MAF_End (page 57) should accurately report success or failure of login operations in the status output parameter:

  Report `0` if the LCM/LSM completes successfully or `non-zero` if the LCM/LSM login operations fail.
- **Don't free memory more than once:** Login Server Modules that call MAF_Free (obsolete 3/1/2006) (page 58) or MAF_MemFree (page 68) more than once to free the same memory might cause eDirectory to crash.
- **Protect global configuration and user secret data:** Data stored in the Login Configuration Store and the Login Secret Store with the method ID of `0` or AIDs of NMAS_AID_USER_GLOBAL_CONFIG_DATA or NMAS_AID_USER_GLOBAL_SECRET_DATA is accessible by all login methods when calling MAF_GetAttribute (page 59) and MAF_PutAttribute (page 72). Data not intended to be shared by multiple login methods should not be stored as Global Login Configuration data and Global Login Secret data . Store that type of data using the method ID assigned to the method or using the AIDs of NMAS_AID_USER_CONFIG_DATA or NMAS_AID_USER_SECRET_DATA.
- **Do not exchange secret data in clear text:** Passwords or other secret data should not be transmitted between the LCM and LSM in the clear when calling MAF_GetPassword (page 61), MAF_GetPasswordEx (page 63), and MAF_SetPassword (page 75). Call MAF_XRead (page 82), MAF_XWrite (page 84), and MAF_XWriteRead (page 85) to encrypt data before it is transmitted and decrypted after it is received.
- **Consider how sensitive data is reported:** Sensitive information, such as passwords, should not be reported to the NMAS audit log when calling MAF_LogEvent (page 65) or to DS Trace when calling MAF_Trace (page 76) or MAF_TraceOnError (page 78).

2.3 NMAS Client

This section is an example of how to integrate the NMAS Client into the Novell Client™. The NMAS Client does not require Novell Client32, and other applications have also integrated with an NMAS Client. A new NMAS Java* Client is also now available.

The NMAS Client must contain at least the following components:

- Section 2.3.1, “Login Dialog Box,” on page 17
- Section 2.3.2, “NMAS Client Manager,” on page 17

- Section 2.3.3, "NMAS on Linux," on page 18
- Login Client Modules (LCM) (see Section 3.5, "Building an NMAS LCM," on page 34).

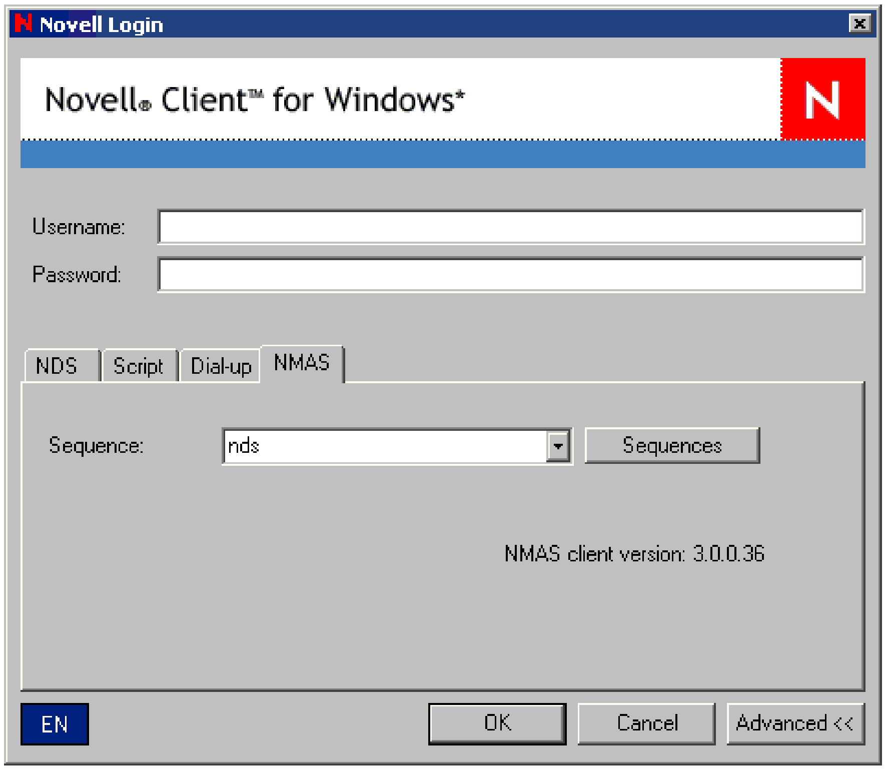
2.3.1 Login Dialog Box

The login dialog box is implemented through the Windows 95/98 and Windows NT* Client login extensions. The master dialog box is changed so that it can be configured to only request the username, because each login method invokes its own GUI dialog box to request the password, PIN, or other module-specific information.

The login dialog box is invoked under the current Client32 events. For Windows NT/2000, the login screen appears when the user has pressed the Ctrl+Alt+Del key sequence.

As shown in Figure 2, the first login dialog box allows the user to select the advanced options window. This dialog box is also modified to include the ability for the user to request a specific login sequence that includes one or more of the login methods. The dialog box also includes the ability for the user to request the Graded Authentication Authorization level (Clearance) by selecting the NMAS tab.

Figure 2-3 *Enhanced Novell Client Login Dialog Box (showing the Sequence field)*



Sequence: nds

NMAS client version: 3.0.0.36

2.3.2 NMAS Client Manager

Client32 invokes the NMAS Client Manager after the login screen is completed.

The NMAS Client manager is responsible for the following:

- Creating an NMAS session for the current session.

- Establishing a session key using the Novell International Cryptographic Infrastructure (NICI) Client.
- Storing the login-gathered data as attributes that can be read by the Login Client Module (LCM).
- Returning the status of Login.
- Determining the available login methods supported by this client.
- Initiating the MAF protocol.
- Receiving the "DO" MAF commands and invoking the appropriate LCM modules (see MAF protocol in Figure 2-2 on page 15 and Figure 2-4 on page 18).
- Upon receipt of the "SUCCESS" MAF command, it receives the credential materials from the server and places them into the authentication store for use by the Authentication Manager.

2.3.3 NMAS on Linux

Linux servers have been running LSMs on eDirectory and NMAS for many years. However, the NMAS client has recently been ported to Linux. Consequently, it is now possible to run NMAS LCMs from a Linux client. This enables you to extend Linux login security by creating a pluggable authentication module (PAM) that invokes your LCM for additional security. To install the NMAS client and create a sample LCM and LSM, see "Tasks for Writing a Login Method" on page 23. For specific information related to Linux, see Section 3.3.4, "Building a Linux LSM," on page 33 and Section 3.5.3, "Building a Linux LCM," on page 35.

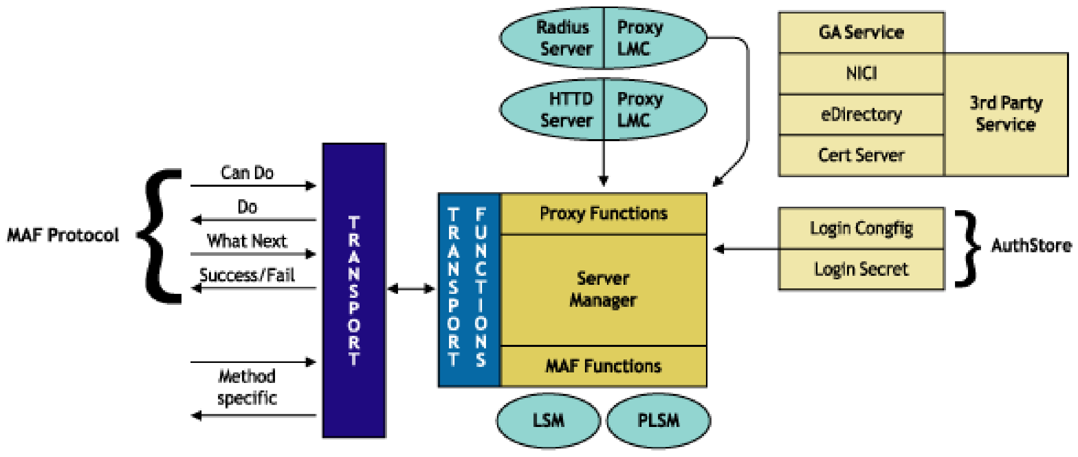
2.4 NMAS Server

The NMAS Server contains the following components:

- Section 2.4.1, "NMAS Server Manager," on page 19
- Section 2.4.2, "Authentication Store," on page 19

Figure 2-4 provides an overview of how the NMAS Server operates:

Figure 2-4 *The NMAS Server and MAF Protocol Architecture*



The NMAS Server provides the server side of the MAF Protocol, which negotiates the login sequence to be used, invokes LSMs, provides an API set that can be used by the LSMs to communicate with the Login Client Modules (LCMs), and provides an API set (MAF functions) to retrieve and store login information in the eDirectory tree. LCMs provide the user interface to prompt the user to supply the information required for the login method. An LCM then passes information to the LSM for verification of the user's identity.

2.4.1 NMAS Server Manager

The NMAS Server Manager is invoked when the NMAS module is loaded on the server. This manager registers with the NCP provider to receive the NMAS NCPs (number 94). This manager also registers with other transports so that it can obtain requests for authentication. Where application-specific protocols are used for login, the NMAS Server Manager provides a set of APIs so that Proxy Services can be implemented.

The Login Manager is responsible for validating the identity of a user so those credentials can be returned to the Client for use in subsequent login calls.

The NMAS Server manager is responsible for the following tasks:

- Creating an NMAS Session for the current session.
- Receiving MAF requests for login.
- Receiving the Initial MAF Login Request from the client.
- Determining the available login methods supported by this Client and selecting the appropriate sequence to be used for this session.
- Sending the "DO" MAF commands and invoking the appropriate Login Server Module modules (see MAF protocol in Figure 2-2 on page 15 and Figure 2-4 on page 18).
- Determining whether the appropriate conditions have been met to log the user in, and if so, building the required credentials and sending them back to the Client with a SUCCESS status.
- Returning the status of the login.

2.4.2 Authentication Store

To support NMAS, an Authentication Store is added to the eDirectory tree, which consists of a set of containers, objects, and attributes. The installation program and ConsoleOne snap-ins create and manage the objects as necessary. Figure 2-5 provides an overview of these components and describes their functionality.

NOTE: NMAS supports only the authentication of users who have User objects in the system. It does not support system access by members of other organizations who have some form of enabling or access credential.

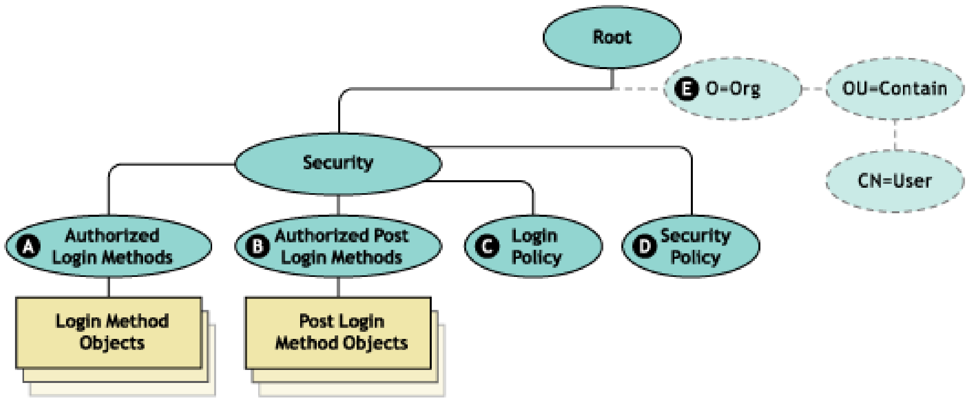
As illustrated in the following diagram, authorization login methods and authorized post login methods are used to store login secrets and login configurations specific to the login method but not specific to a user. The Authorized Login Methods container holds the login method objects. There is a login method object for each login method.

A post login method (PLMO) has an LCM, an LSM, and usually a Method Management Graphical Interface (MMG). A PLMO contains the signed executable code for the LSMs. Therefore, a method only needs to be installed once for a tree, not for every server. This simplifies distribution of login methods.

LSM executable code and method login-specific login secret and login configuration information are stored in the login method objects. The Login Policy object contains login policy information, for example, login sequences. A login sequence is an ordered list of login methods and post login methods.

The Security Policy defines a Graded Authentication policy that include categories, labels, clearances, eDirectory attributes, etc. The User Object has a login secret and login configuration that is specific to a user and a login method.

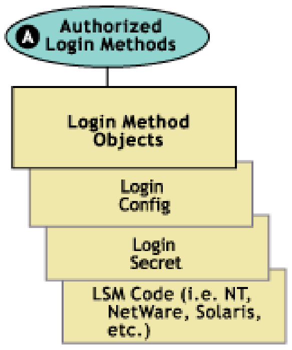
Figure 2-5 *Authorized Login Method Architecture*



Authorized Login Method Container (LMC)

Figure 2-6 shows the Login Method container (LMC) containing the Login Method objects (LMOs). There is an LMO for each of the login methods that have been installed into the tree. An LMO contains the signed executable code for the LSMs. LSM code also is provided for the various platforms (Windows, NetWare, Solaris*, etc.).

Figure 2-6 *Authorized Login Method Container (LCM)*


A Authorized Login Methods
Login Method Objects
Login Config
Login Secret
LSM Code (i.e. NT, NetWare, Solaris, etc.)

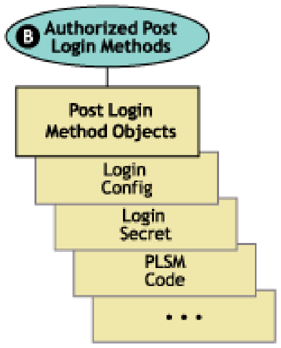
The LMO also can be used to store login secrets and login configuration data that is specific to the login method but not specific to a user. A Login Device object (LD) is specific to an LMO and is neither managed or created by NMAS. If used, LDs are created and managed by a login method's MMG.

NOTE: The maximum size of the login configuration and login secret data is 60,000 bytes.

Authorized Post Login Methods (APLM) Container

Figure 2-7 shows the Authorized Post Login Method container that contains Post Login Method objects (PLMO). There is a PLMO for each post login method installed into the tree.

Figure 2-7 *Authorized Post Login Methods (APLM) Container*

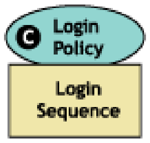


The main difference between login methods and post login methods is that post login methods are invoked after login has been completed. Examples are the Screen Saver and Change Password.

Login Policy

Figure 2-8 shows the Login Policy Object, which contains login sequence definitions. This defines which login methods are required to authenticate to the network. The login policy is stored within the eDirectory security container. This attribute is "public read," which means that a connection can read the available login sequences whether it is authenticated or not.

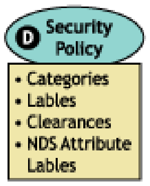
Figure 2-8 *Login Policy Object*



Security Policy Container

Figure 2-9 shows the Security Policy Object, which defines the categories, labels, and signed eDirectory attribute labels. This object holds the tree-wide graded authentication policy. All servers in the tree reference the common policy.

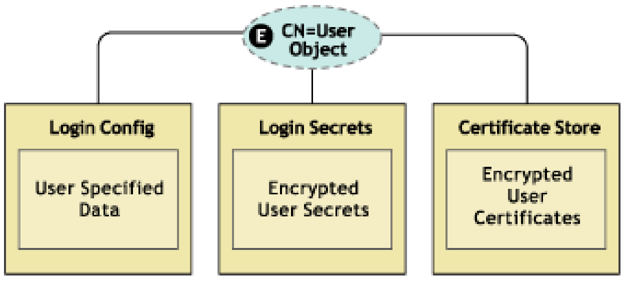
Figure 2-9 *Security Policy Object*



Config Store/Secret Store Attributes

NMAS allows the creation of Config Store and Secret Store Attributes on the LMO, PLMO, and User objects. These are shown in Figure 2-6, Figure 2-7, and Figure 2-10, respectively. The data contained in these attributes are tagged so that a specific method can access the data from a Login Server Module (LSM).

Figure 2-10 *NMAS User Login Storage Information.*



We recommend that you use the Login Method object and Login Device object config/secret store attributes for static data and the User object config/secret store attributes for dynamic data.

NOTE: The security container is designed to be highly replicated to almost all servers in the tree. For this reason, all objects under the security container should contain relatively static data. This means that data stored in PLMOs, LMOs, and LDOs should not be updated on each login. If dynamic data is desired, you should place this on the user's object or create an object outside the security container that is not globally replicated.

3 Tasks for Writing a Login Method

The NMAS™ SDK allows you to develop a NMAS login method that implements your own login system using the NMAS framework. As a developer to NMAS, you build a login method to pass credentials and authenticate to the network through eDirectory™.

Essentially, each login method consists of a Login Client Module (LCM), a Login Server Module (LSM), and a Method Manager Graphic Interface (MMG) for administrating and installing your method. To understand the interrelationship of these components, see the Section 2.2, "Login Method Overview," on page 14.

NOTE: Ensure that the LSM and LCM are threadsafe.

To preserve the integrity of the login process, only login methods signed by Novell® can be implemented into the NMAS framework. Before building your login method, contact Novell to register your method and obtain a unique method number.

As part of this registration process, use the signing tools from the NMAS component (http://developer.novell.com/wiki/index.php/Novell_Modular_Authentication_Service) of the Novell Developer Kit (NDK) to generate a certificate signing request. To get your method signed and obtain a unique certificate, contact Kamal Narayan, NMAS Product Manager (mailto:nkamal@novell.com).

NOTE: Login method development can begin before a method number is assigned using the test method number 1 and the testing procedures described in Section 3.7, "Testing Your NMAS Methods," on page 40.

This section contains the following topics:

- Section 3.1, "Building an NMAS Login Method," on page 23
- Section 3.2, "Generating a Method Signing Certificate Request," on page 24
- Section 3.3, "Testing-Debugging Unsigned Login Methods," on page 31
- Section 3.4, "Installing the Clear Text Password Method," on page 34
- Section 3.5, "Building an NMAS LCM," on page 34
- Section 3.6, "Enrolling a User with the Clear Text Password," on page 36
- Section 3.7, "Testing Your NMAS Methods," on page 40

3.1 Building an NMAS Login Method

To build a NMAS login method, you need to complete these basic tasks, which are described in the following sections:

1. Generate a Method Signing Certificate.
2. Sign Your NMAS Login Method.

3. Build a Login Server Module (LSM).
   - Section 3.3.2, "Building a Windows LSM," on page 32.
   - Section 3.3.3, "Building a NetWare LSM," on page 32.
   - Section 3.3.4, "Building a Linux LSM," on page 33.
4. Build a Login Client Method (LCM).
   - Section 3.5.2, "Building a Windows LCM," on page 35.
   - Section 3.5.3, "Building a Linux LCM," on page 35.
   - Section 3.5.4, "Building a Java LCM," on page 36.
5. Enroll the user with the new password.
6. Test the new method. Complete Novell Yes CertifiedTM Program testing and ensure that your method is enabled for eDirectory. Complete the testing and send the method to Novell or an authorized Novell testing lab for verification. For more information, see Section 3.7, "Testing Your NMAS Methods," on page 40.

3.2 Generating a Method Signing Certificate Request

A Login Server Module (LSM) must be signed to operate under the NMAS framework. To sign a method, you must use the signing tools located in the `c:\novell\ndk\`*`nmas ndk`*`\nmas\sign` directory of the NDK download for NMAS (see NMAS Developer page (http://developer.novell.com/wiki/index.php/Novell_Modular_Authentication_Service)).

The signing kit consists of a set of batch files that run the appropriate signing programs. These batch files generate module signing key pairs that are used to request a method certificate from Novell. The keys also are used to sign your modules.

You need to send your certificate request to Novell, so that a certificate signed for use with the NMAS signing kit can be produced by Novell and returned to you with your assigned Method ID.

You must have a client Novell International Cryptography Infrastructure (NICI) installed on a WIN95/98/NT/2000 or workstation to use the signing tools (see the NICI download page (http://www.novell.com/products/cryptography)).

IMPORTANT: The signing toolkit requires NICI 1.5.7 The signing kit contains both BAT files and EXE files that are executed from a DOS window on a Windows 95/98/NT/2000 platform. You should only use the BAT files when using the signing kit.

This section contains the following topics:

- Section 3.2.1, "Steps for Generating a Signing Key," on page 25
- Section 3.2.2, "Steps for Signing an LSM," on page 26
- Section 3.2.3, "Packaging an NMAS Method," on page 27
- Section 3.2.4, "Novell Yes CertifiedTM Program," on page 31

3.2.1 Steps for Generating a Signing Key

Generating your signing keys is a one-time process that must be done for each method developed for NMAS:

1 Copy all files from the Method Signing Kit to the directory that is unique to each method. For example: `c:\nmas\tools\methods\`*`xx`* where *xx* is a unique name for your method.

You must use a separate directory for each method. Execute all configuration and signing operations from this directory.

2 Generate a method signing key pair by running `keygen.bat`.

2a You are prompted for your Vendor Name and a password. The Vendor Name is displayed by NMAS when this method is loaded. The password is used to protect your module signing private key on your system. Use the password to complete the certificate request each time your method is signed.

2b This utility generates two files: `private.p7b`, your encrypted private key; and `self.ber`, your self-signed certificate.

IMPORTANT: Protect your private key appropriately, because your certificate verifies the modules signed with the private key with your corporate name.

For example:

```
>cd \nmas\tools\methods\xx\
>keygen
Enter Vendor Name: ACME, Inc.
Password: ********
Retype Password: ********
```

3 Generate a certification request for your public key by running `cr.bat`.

3a You must specify an authentication grade by providing the hexadecimal value as the first argument to cr.bat. You can use the following hex values for the grades supported by NMAS:

Grade	Hex Value
Logged In	0x00000000
Biometric	0x00200000
Password	0x00800000
Token	0x00400000
Biometric and Password	0x00A00000
Biometric and Token	0x00600000
Password and Token	0x00C00000
Biometric and Password and Token	0x00E00000

NOTE: Novell reserves the right to change the method grade when processing the certificate signing request.

3b You are prompted for your password to decrypt your stored private key that is used to sign your request. This utility generates one file, `csr.ber`, per certification request.

Example for a Biometric and Token method request:

```
>cr 0x00600000Password: ********
```

3c Send the file csr.ber to Novell for further information. For more information, see the NMAS Login Method Registration Form (http://developer.novell.com/devres/ss/nmasform.htm).

3d You will receive two files from Novell after the certification process: `cert.ber` (your signed certificate) and `mib.ber` (your module identification block).

3e Copy these two files to your module-signing directory as defined in Step 1. These two files are used during the module signing process described in "Steps for Signing an LSM" on page 26.

3.2.2 Steps for Signing an LSM

Use the following procedure to build and sign an LSM that is named `LSMCPWD.NLM.LMO`:

1 Code and build the NLM.

On NetWare:

Import the following symbols:

```
MAF_Begin, MAF_End, MAF_Read, MAF_Write, MAF_WriteRead, MAF_XRead, MAF_XWrite,
MAF_XWriteRead, MAF_PutAttribute, MAF_Get Attribute
```

The `makefile.mak` file is provided to assist in constructing your build environment.

On Windows NT:

Link against nmas.lib and include the NMAS header files. The `lsmcpwd.dsw` and `lsmcpwd.dsp` project files are provided as a reference to assist you in constructing your build environment.

Export the entry points in your module that are called by the NMAS loader during the login process. It is conventional that the method be included in the entry point name. For example, if the method number is 9, the entry point name could be LSM00000001 or LSM1.

2 Place the NLM and/or DLL that contains your LSM in the method-signing directory. This is the directory that contains the signing certificate created in Section 3.2, "Generating a Method Signing Certificate Request," on page 24.

3 Run `sign.bat` to sign the method for execution under NMAS.

3a You must give three arguments to this utility. The first one is the name of the file you want to sign. The second is the name of your method as provided when the NLM was linked. The third argument is the name of the entry point in your module that is called by the NMAS loader.

3b When prompted, provide your password to be used to decrypt the private key and sign the module (for example, LSM00000001). This utility generates one file: your signed module with the LMO extension in your method-signing directory.

For example, assuming your module is named `LSMCPWD`, you gave the NLM the name `NLMNAME` when it was linked, and the entry point is LSM00000001, the following command creates the file `LSMCPWD.NLM.LMO` that is a signed method that can be installed into NMAS:

```
>sign LSMCPWD.NLM LSMCPWD.NLM LSM00000001
```

3c Place the signed method in the appropriate directory to install into NMAS.

3.2.3 Packaging an NMAS Method

NMAS methods contain both client and server components. Normally, you install an NMAS client component as a part of its own product install. The server NMAS components, however, can be installed using the ConsoleOne® create login method snap-in, as described in "Method Creation" on page 27.

Client Packaging

Client NMAS method packaging is usually done as part of a developer's normal product install. This should provide the customer with a more integrated installation, because the NMAS components can be installed at the same time as the developer's hardware/device components of the method. The client install should include the following for NMAS:

1. Place the `LCM DLL` in the Windows SYSTEM directory. (WinNT/2000 is Winnt/System32. Win95/98/ME is the Windows/System.)
2. Update the registry setting as described in Step 3 on page 35.

Server Packaging

Methods are packaged into individual directories so that they can be easily distributed to customers and installed into the tree. The recommended convention is to place the methods under the following directory scheme:

`\nmasMethods\`*`Your Company Name`*`\`*`MethodName`*

where `MethodName` is the method root directory of all files related to this method.

Method Creation

The `config.txt` file, which provides the information needed for the NMAS create login method snapin to install the method, is placed under the method root directory. The following directories and files are also placed in the method root directory.

`\client`	A directory that contains the LCM module with the appropriate installation procedures and instructions.
`\ConsoleOne\resources\Security`	The directory that contains the method-specific MMG resource jars.
`\ConsoleOne\snapins\Security`	The directory that contains the method-specific MMG snap-in jars.
`\setup\<lang>`	Directories that contain text and graphics that might be internationalized. The lang directory follows the Java locale format so that the method can support multiple languages.
`config.txt`	The LSM installation and configuration file. This file is described in "The config.txt File" on page 28.
`YOURMETHOD.LMO`	The LSM/LCM modules that have been signed for use with NMAS.
`SCHEMA.sch`	An optional file that is used to extend the eDirectory schema when the method is installed.
`readme.txt`	An optional file that contains information about this method.

The config.txt File

The installation process for a new login method is driven by a simple configuration file. The name of this file can be any desired name so long as the file extension is .txt. The suggested name for this file is config.txt. This file can be created with a simple text editor. The format of the configuration file is:

```
Parameter = Value
```

Each entry in the configuration file is limited to a single line. Parameter values are not case-sensitive. The recognized parameters include the following:

Parameter	Description of Value
Name (required)	The name of the method. This name is used by default as the name of the login method object (LMO) in Novell Directory Services (NDS®) eDirectory. This name can be changed if desired during the installation.
Vendor (required)	The name of the vendor of the login method. This name cannot be changed during the installation process. This value is stored as an attribute of the login method object (LMO) in eDirectory.
Grade (required)	The value of this parameter is one or more of the following tokens: ◆ Biometric ◆ Token ◆ Password ◆ Logged in To use more than one of these tokens, the ampersand sign (“&”) is used between them. For example, if a method includes both token and password grades the value would be Token & Password. This value is advisory only. The actual grade accorded to the method is found in the Login Server Module (LSM) for the method. The advisory grade is stored as an attribute of the login method object (LMO) in eDirectory.
Method ID (required)	This is the Method ID assigned by Novell to the vendor and uniquely identifies this method. The value can be specified in decimal (for example, 15) or hexadecimal (for example, 0x0f) or octal (for example, O17). The value of this parameter is advisory only. The actual value is found in the Login Server Module (LSM) for this method. The Method ID is stored as an attribute of the login method object (LMO) in eDirectory.
Method Type (optional)	Specifies the type of method. Currently, the only valid value is Certificate.
Description file (optional, localized)	Specifies the name of the file that holds a brief description of the login method. The content of this file is stored as an attribute of the LMO. The file is located in `\setup\Language\description_file_name`.
License file (optional, localized)	Specifies the name of the file that holds the text of the vendor’s license. This license is presented to the user and acceptance of the license is required before the method can be installed into the directory. The license file must be a plain text file. For best viewing no line in the license file should be longer than 60 characters. This file is not stored in the directory. The file is located in `\setup\Language\license_file_name`.

Parameter	Description of Value
Support file (optional, localized)	Specifies the name of the file that holds information about how to obtain support for the login method. This could include vendor phone numbers, a URL on the Web, or other support information. The content of this file is stored as an attribute of the login method object (LMO) in eDirectory. The file is located in `\setup\Language\support_file_name`.
Logo file (optional, localized)	This file contains the logo (or logos) of the companies that contributed technology to this login method. The logo file must be in GIF format. The maximum size for a logo is 345 pixels wide and 175 pixels tall. The logos are displayed on the last page of the create login method sequence just before the login method object (LMO) is created. Although this file is a "localized" file, the vendor can choose to supply only one logo and have it used for all languages. This can be done by placing this file only in the setup\en (English) directory. The file is located in `\setup\Language\logo_file_name`.
Schema file (optional, not localized)	If the login method requires the eDirectory schema to be extended, then this parameter specifies the name of the schema extension file (SCH file). During installation, this file is processed to extend the eDirectory schema as required.
LSM NetWare (optional, not localized)	Specifies the name of the file containing the code of the Login Server Module (LSM) for NetWare. The contents of the file are stored in the directory.
LCM NetWare (optional, not localized)	Specifies the name of the file containing the code of the Login Client Module (LCM) for NetWare. The contents of the file are stored in the directory.
LSM WinNT (optional, not localized)	Specifies the name of the file containing the code of the Login Server Module (LSM) for Windows NT. The contents of the file are stored in the directory.
LSM Win_X64 (optional, not localized)	Specifies the name of the file containing the code of the Login Server Module (LSM) for Windows 64-bit. The contents of the file are stored in the directory.
LCM WinNT (optional, not localized)	Specifies the name of the file containing the code of the Login Client Module (LCM) for Windows NT. The contents of the file are stored in the directory.
LCM Win_X64 (optional, not localized)	Specifies the name of the file containing the code of the Login Client Module (LCM) for Windows 64-bit. The contents of the file are stored in the directory.
LSM Solaris (optional, not localized)	Specifies the name of the file containing the code of the Login Server Module (LSM) for Solaris. The contents of the file are stored in the directory.
LSM Solaris_64 (optional, not localized)	Specifies the name of the file containing the code of the Login Server Module (LSM) for Solaris 64-bit. The contents of the file are stored in the directory.
LCM Solaris (optional, not localized)	Specifies the name of the file containing the code of the Login Client Module (LCM) for Solaris. The contents of the file are stored in the directory.
LCM Solaris_64 (optional, not localized)	Specifies the name of the file containing the code of the Login Client Module (LCM) for Solaris 64-bit. The contents of the file are stored in the directory.
LSM Linux (optional, not localized)	Specifies the name of the file containing the code of the Login Server Module (LSM) for Linux. The contents of the file are stored in the directory.
LSM Linux_X64 (optional, not localized)	Specifies the name of the file containing the code of the Login Server Module (LSM) for Linux 64-bit. The contents of the file are stored in the directory.
LCM Linux (optional, not localized)	Specifies the name of the file containing the code of the Login Client Module (LCM) for Linux. The contents of the file are stored in the directory.
LCM Linux_X64 (optional, not localized)	Specifies the name of the file containing the code of the Login Client Module (LCM) for Linux 64-bit. The contents of the file are stored in the directory.

Parameter	Description of Value
LCM AIX (optional, not localized)	Specifies the name of the file containing the code of the Login Client Module (LCM) for AIX*. The contents of the file are stored in the directory.
LSM Tru64 (optional, not localized)	Specifies the name of the file containing the code of the Login Server Module (LSM) for Tru64. The contents of the file are stored in the directory.

Placement of Files

1 Place the `LCM` and `LSM LMO` files in the same directory as the `config.txt` file.

2 Place the translated versions of the description file, the license file, and the support file in the following directories:

Language	Directory Location
Czech	cs_CZ
German	setup/de
English	`setup/en`
Spanish	setup/es
French	setup/fr
Italian	setup/it
Japanese	setup/ja
Dutch	setup/nl_NL
Polish	setup/pa_PL
Portuguese	setup/pt
Russian	setup/ru

3 Install the ConsoleOne snapin or the NMAS Method Install iManager plugin (http://www.novell.com/documentation/nmas311/admin/data/a49tuwk.html#b5il5e8).

Schema Modifications

Normally, the NMAS methods can be implemented without modifications to the schema. However, if additional objects and attributes are desired, you can create a .sch file that contains your schema definitions. Samples of these files are found on NetWare servers in the `SYS:SYSTEM\SCHEMA` directory. The NMAS Create Login Method object can be configured to extend the schema for you.

Partners who are extending the eDirectory schema should register their extension with Novell Developer Support (http://developer.novell.com/support).

The NMAS code that extends the schema during creation of a login method requires the schema elements (attributes and classes) to have assigned ASN1 identifiers that are assigned through the schema registration process.

3.2.4 Novell Yes Certified™ Program

As part of the developing your method to operate with NMAS, you should give consideration to having your method Yes Certified™. Products that have successfully completed all testing and business requirements are identified and can be marketed with a Yes Certified logo.

For more information about Yes CertifiedD, refer to the following URL:

http://developer.novell.com/devnet/yes/page7.html (http://developer.novell.com/devnet/yes/page7.html)

For developer support tools, refer to:

http://developer.novell.com/devres/ss (http://developer.novell.com/devres/ss)

3.3 Testing-Debugging Unsigned Login Methods

The NMAS product ships with a number of Novell-supported methods and the NMAS SDK (http://developer.novell.com/wiki/index.php/Novell_Modular_Authentication_Service) includes a clear text password method and other NMAS-related samples. (See the NMAS Sample Code and Demo Applications (http://developer.novell.com/wiki/index.php/Novell_Modular_Authentication_Service_Samples).)

Because this sample method has not be signed using the signing kit mentioned above, it is necessary to test it using the debug version of NMAS. The debug version of NMAS expects to find the LSM on the file system as opposed to the release version of NMAS, which looks for the LSM in eDirectory stored as an Login Method Object (LMO).

IMPORTANT: In order for these samples to work, you must first download and install the NLM and NetWare Libraries for C (http://developer.novell.com/wiki/index.php/NLM_and_NetWare_Libraries_for_C_%28including_CLIB_and_XPlat%29). Project files and makefiles expect the CLIB NDK at `c:\novell\ndk\clib` on Windows and `/opt/novell/ndk/clib` on Linux.

To build a Login Server Module (LSM), follow the procedures outlined in:

- Section 3.3.1, "Requirements for Building an LSM," on page 31
- Section 3.2.2, "Steps for Signing an LSM," on page 26
- Section 3.2.3, "Packaging an NMAS Method," on page 27
- Section 3.2.4, "Novell Yes CertifiedTM Program," on page 31

3.3.1 Requirements for Building an LSM

The following are the requirements for building an LSM:

- A Development environment (Visual Studio, Eclipse, Watcom, Metroworks, etc.) and the Novell Developer Kit (NDK). For more information, see Section 1.2, "Selecting a Compiler," on page 11.
- The file `maf.h`, located in current nmas ndk\nmas_server_sdk\sdkinc.
- The file `nmasapi.h`, located in current nmas ndk\nmas_server_sdk\sdkinc.

- The file `nmaserr.h`, located in current nmas ndk\nmas_server_sdk\sdkinc
- A Signing Certificate from Novell as requested in Section 3.2, "Generating a Method Signing Certificate Request," on page 24.

3.3.2 Building a Windows LSM

1 Build the LSM by executing the `build.bat`

```
c:\novell\ndk\Current NMAS
NDK\nmas_sample_code\login_method\lsm\win32\src\build.bat
```

WARNING: The `nmas.lib` that links with the LSM is NOT the same `nmas.lib` that links with the LCM.

2 Create the directory structure `c:\novell\nds\nmas\lsm` on Windows NT/2000/XP.

NOTE: The directory nmas\lsm is not there by default.

3 Create the `c:\novell\nds\nmas\lsm\idlist.txt` file that contains the following line:

```
1 lsmcpwd.dll LSM00000001
```

NOTE: The first field `idlist.txt` is the method ID and the method ID should be represented as a hex number.***

4 Rename `c:\novell\nds\nmas.dlm` to `nmas.dlm.rel` (save the release version).

5 Add `C:\Novell\NDS` to your Path environment variable.

6 Copy the `C:\Novell\ndk\`*`nmas ndk`*`/\nmas_server_sdk\win32\bin\debug\nmas.dlm` to `c:\novell\nds`.

7 Install the LSM (see Section 3.4, "Installing the Clear Text Password Method," on page 34).

8 Restart the eDirectory service.

8a Select Novell eDirectory Services from the Control Panels Window or the Desktop.

8b Press the Shutdown button and wait until this button is disabled

8c Click the Startup button.

TIP: You can also stop and start the eDirectory service using the control panel window located in the services icon of the eDirectory Administrative Tools.

3.3.3 Building a NetWare LSM

For NetWare development, use either the MetroWerks CodeWarrior compiler (available with the Novell Software Evaluation Library (http://developer.novell.com/brochure)) or the free Watcom 11.0c compiler (http://www.openwatcom.org) from a Windows system.

1 Build the LSM by executing the following `.bat` files in the order indicated:

`c:\novell\ndk\`*`nmas ndk`*`\nmas_sample_code\login_method\lsm\nw\`*`metrowerks`*`\env.bat`

`c:\novell\ndk\`*`nmas ndk`*`\nmas_sample_code\login_method\lsm\nw\`*`metrowerks`*`\build.bat`

NOTE: If using the *watcom* compiler, replace *metrowerks* in the above path with *watcom*.

2 Create the following directory structure required by the debug version of NMAS

```
sys:\system\nmas\lsm
```

3 Copy the LSM to this debug directory

```
copy c:\novell\ndk\nmas ndk\nmas_sample_code\login_method\lsm\nw\debug
sys:\system\nmas\lsm
```

4 Create the `sys:\system\nmas\lsm\idlist.txt` file required by the debug version of NMAS that contains the following line:
1 lsmcpwd.nlm LSM00000001

5 Save the release version of NMAS and switch to the debug version

```
ren sys:\system\nmas.nlm sys:\system\nmas.nlm.rel
copy c:\novell\ndk\nmas ndk\nmas_server_sdk\nw\bin\debug\nmasdbg.nlm
sys:\system\nmas.nlm
```

6 Install the LSM (see Section 3.4, "Installing the Clear Text Password Method," on page 34).

7 Restart NetWare by executing

```
restart server
```

3.3.4 Building a Linux LSM

1 Compile the LSM by executing the Linux.mak from the debug directory

```
cd /opt/novell/ndk/nmas ndk/sample/methodExample/lsm/unix/Linux/debugmake -f
../Linux.mak
```

2 Create the following directory structure required by the debug version of NMAS

```
/var/nds/nmas-methods/NMAS/LSM
```

NOTE: The directory /var/nds/nmas-methods only exists for eDirectory 8.7.3.*x*. For eDirectory, 8.8 the directory (by default) is `/var/opt/novell/eDirectory/data/nmas-methods`.***

3 Copy the LSM to this debug directory

```
cp /opt/novell/ndk/nmas ndk/nmas_sample_code/login_method/lsm/unix/Linux/
debug/lsmcpwd.so /var/nds/nmas-methods/NMAS/LSM
```

4 Create the /var/nds/nmas-methods/NMAS/LSM /IDLIST.TXT required by the debug version of NMAS with the following line:
1 lsmcpwd.so LSM00000001

5 Save the release version of NMAS and switch to the debug version

```
mv /usr/lib/nds-modules/libnmas.so /usr/lib/nds-modules/libnmas.so.relcp /opt/
novell/ndk/nmas ndk/nmas/nmas_server_sdk/linux/bin/debug/libnmas.so /usr/lib/
nds-modules
```

NOTE: The directory `/usr/lib/nds-modules` is for eDirectory 8.7.3.*x*, while the corresponding directory for eDirectory 8.8 is `/opt/novell/eDirectory/lib/nds-modules`.***

6 Install the LSM (see Section 3.4, "Installing the Clear Text Password Method," on page 34).

7 Restart eDirectory

```
/etc/init.d/ndsd restart
```

3.3.5 NMAS NDK Sample Code

The NMAS Sample Code (http://developer.novell.com/ndk/doc/samplecode/nmas_sample/index.htm) demonstrates how to implement many of the API functions described in this document.

3.4 Installing the Clear Text Password Method

1 Create a `..\ndk\nmas ndk\nmas_sample_code\login_method\config.txt` file used to install the method that contains the following:

```
name=lsmcpwd
Vendor=Novell,Inc.
grade=Logged in
methodid =1
```

NOTE: ***The value of the name field can only contain the following characters:

`abcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ.&-!#$%()*[]^_~`

---

2 Using ConsoleOne, install the method.

2a On a Windows console, launch ConsoleOne from the icon located on the desktop.

2b On a Linux console, launch ConsoleOne by executing the following command:
`/usr/ConsoleOne/bin/ConsoleOne`

2c Select the NDS® object and then click on the tree in the toolbar and enter the following login credentials:
Login name: admin
Password: *admin's NDS password*
Tree: 127.0.0.1
Context: novell

2d From ConsoleOne, expand the NDS tree including the Security container object. Right click on the Authorized Login Methods container, then select New, then click Object (SAS:NMAS Login Method), and then click OK.

2e From the New Login Method Window, use the Browse button to select the `c:\novell\ndk\`*`Current NMAS NDK`*`\nmas_sample_code\login_method\config.txt` file and click the Open button.

2f Follow the prompts and select all of the default settings.

3 On Linux you can also install the method using `nmasinst`.
Format:
nmasinst -addmethod <*admin.context*> <*treeName*> <*configFile*>
Example:
nmasinst -addmethod admin.novell corp-tree ./config.txt

3.5 Building an NMAS LCM

To build a Login Client Module (LCM), follow the procedures outlined in:

- Section 3.5.1, "Requirements for Building an LCM," on page 35
- Section 3.5.2, "Building a Windows LCM," on page 35
- Section 3.5.3, "Building a Linux LCM," on page 35
- Section 3.5.4, "Building a Java LCM," on page 36

3.5.1 Requirements for Building an LCM

The following are the requirements for building an LCM:

- An development environment such as Microsoft Visual C++ Version 6 or later, Eclipse, etc. and the Novell Developer Kit (NDK).
- The file `maf.h`, located in *Current NMAS NDK*\nmas_client_sdk\sdkinc\legacy
- The file `nmasapi.h`, located in Current NMAS NDK\nmas_client_sdk\sdkinc\
- The file `nmaserr.h`, located in Current NMAS NDK\nmas_client_sdk\sdkinc\

3.5.2 Building a Windows LCM

1 Build the LCM by executing the `build.bat`
```
c:\novell\ndk\Current NMAS
NDK\nmas_sample_code\login_method\lcm\win32\build.bat
```

WARNING: The nmas.lib that links with the LCM is NOT the same `nmas.lib` that links with the LSM.

---

2 Copy the `cpasswd.dll` to the c:\Windows\System32 directory
```
copy c:\novell\ndk\Current NMAS
NDK\nmas_sample_code\login_method\lcm\win32\cpasswd.dll c:\Window\System32
```

3 Register the LCM with the Windows Registry by adding the following String Value entry
```
[HKEY_LOCAL_MACHINE\SOFTWARE\Novell\NMAS\1.0\LCM
Paths]"00000001"="C:\\WINDOWS\\System32\\cpasswd.dll"
```

3.5.3 Building a Linux LCM

1 Compile the LCM by executing the Linux.mak from the debug directory.
```
cd /opt/novell/ndk/nmas ndk/nmas_sample_code/login_method/lcm/linux/debug make
-f ../Linux.mak
```

2 Copy the LCM shared object library to the `/usr/lib` directory.
```
cp /opt/novell/ndk/nmas ndk/nmas_sample_code/login_method/lcm/linux/debug/
libcpwdlcm.so /usr/lib
```

3 Execute build.sh to create a GTK sharp password prompt window
```
/opt/novell/ndk/nmas ndk/nmas_sample_code/login_method/lcm/linux/cpwdgui/
build.sh
```

4 Create following directory structure used by the clear text password LCM for the popup window.
```
/opt/novell/nmas/methods/clrpwd
```

5 Copy the following files from /opt/novell/ndk/*nmas ndk*/sample/methodExample/lcm/linux/cpwdgui to /opt/novell/nmas/methods/clrpwd
```
cpwdguicpwdgui.gladeNmas_simple.bmp
```

6 Install the Linux NMAS Client located at `/opt/novell/ndk/nmas ndk/linux` by executing:
```
rpm ivh /opt/novell/ndk/nmas ndk/linux/bin/novell-nmasclient.i386.rpm
```

This step installs the NMAS Client at /opt/novell/nmas/client/ and modifies the /etc/ld.so.conf file to include /opt/novell/nmas/client/lib in the ldconfig path

7 Create an /etc/nmasclnt.conf file by executing the following commands:

```
/opt/novell/nmas/client/bin/ncc -c create/opt/novell/nmas/client/bin/ncc -ma 1
module=libcpwdlcm.so network_func=LCM00000001
```

8 Compare your `/etc/nmasclnt.conf` with the following:

```
##############################################################
Section: Configuration File Information
##############################################################
[config_info]
version = 2
modification_date = Apr 21, 2005
[/config_info]

##############################################################
Section: NMAS Client Information
##############################################################
[client_info]
version = 3.1.0.4
build_date = Apr 1, 2005
description = NMAS Client, Linux
[/client_info]

##############################################################
cd ..cpw# Section: List of Methods
##############################################################
[method_list]

[method]
method_ID = 1
module = libcpwdlcm.so
network_func = LCM00000001
[/method]

[/method_list]
```

3.5.4 Building a Java LCM

1 Build the Java LCM by executing

`/opt/novell/ndk/`*`nmas ndk`*`/nmas_sample_code/login_method/lcm/java/build.sh`

3.6 Enrolling a User with the Clear Text Password

After creating and installing the clear text password LSM and LCM and installing the method, you must assign the clear text password to a user object. The following sections discuss this topic:

- Section 3.6.1, “Using ConsoleOne on Windows,” on page 37
- Section 3.6.2, “Using Method Management APIs on Linux,” on page 37

3.6.1 Using ConsoleOne on Windows

Using ConsoleOne with the clear text password snap-in is the easiest way to assign the clear text password to a user object. However, all NMAS snap-ins for ConsoleOne only work on Windows because they rely on `NMASWrap.dll`. Use the following procedure:

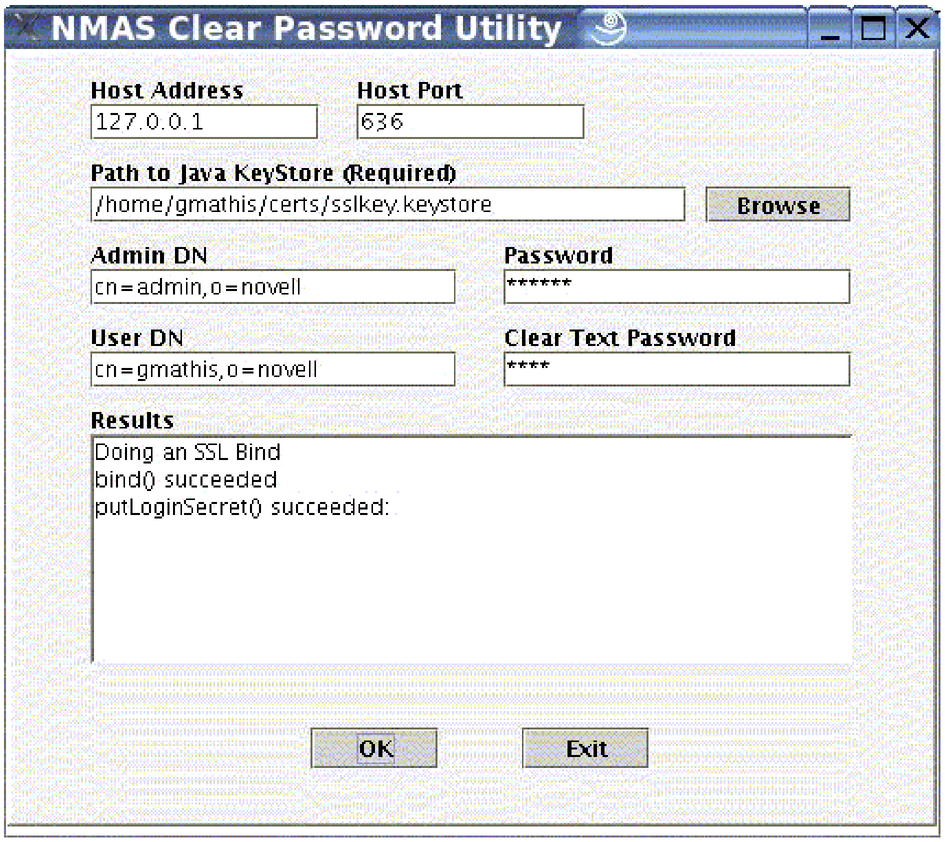
1. Install the latest version of ConsoleOne on a Windows NT/2000/XP system. By default ConsoleOne is installed at c:\novell\consoleone on a Windows system.
2. Install a Java JDK.
3. Compile the Clear Text Password ConsoleOne snap-in by executing the `build.bat` file
   ```
   c:\novell\ndk\nmas ndk\nmas_sample_code\login_method\mgmt\consoleone\build.bat
   ```
4. Copy the `cpwd.jar` to the ConsoleOne snap-ins directory
   ```
   copy c:\novell\ndk\nmas
   ndk\nmas_sample_code\login_method\mgmt\consoleone\cpwd.jar
   c:\novell\consoleone\1.2\snapins\Security
   ```
5. `Restart` ConsoleOne.

3.6.2 Using Method Management APIs on Linux

For a cross platform solution, you will have to run an application that uses the NMAS Method Management APIs. These APIs require you to establish an SSL connection to the server using the server's Trusted Root Certificate. These APIs are available for both C and Java.

Figure 3-1 on page 38 shows a Java sample application (SetClrPwdApp) that uses the Method Management APIs to set a user's clear text password. For Java, these APIs are included in the `NMASToolkit.jar` file.

Figure 3-1 The NMAS Clear Password Utility GUI



Host Address: 127.0.0.1
Host Port: 636
Path to Java KeyStore (Required): /home/gmathis/certs/sslkey.keystore
Admin DN: cn=admin,o=novell
Password: ******
User DN: cn=gmathis,o=novell
Clear Text Password: ****
Results:

```
Doing an SSL Bind
bind() succeeded
putLoginSecret() succeeded:
```

The next few steps demonstrate how to export a Trusted Root Certificate from eDirectory and store it in a Java key store used by the sample application:

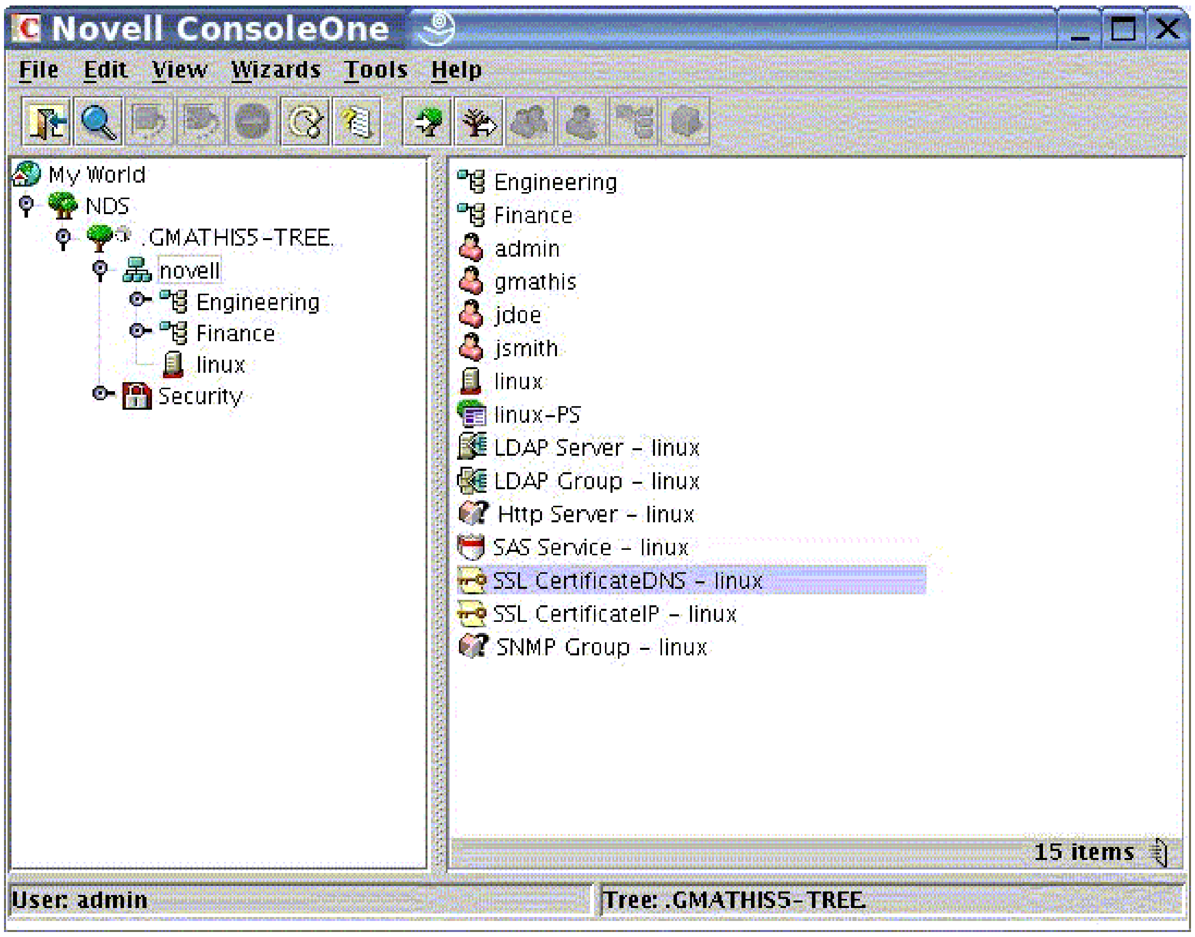
- "Using ConsoleOne to Export the Trusted Root Certificate" on page 38
- "Creating a New User in eDirectory" on page 39
- "Adding a Trusted Root Certificate to a Sun Keystore" on page 40

Using ConsoleOne to Export the Trusted Root Certificate

1 On Linux start ConsoleOne by executing the following:
`/usr/ConsoleOne/bin/ConsoleOne`

2 Select the NDS object and then click on the tree in the toolbar and enter the following login credentials:
Login name: admin
Password: <*admin's NDS password*>
Tree: 127.0.0.1
Context: *novell*

Figure 3-2 *Novell ConsoleOne GUI*



3 Expand the NDS tree and select the organization where the server object is located.

3a Select the properties of the "SSL CertificateDNS - linux object", and then click the Certificate tab.

3b Make sure Trusted Root Certificate is selected on the tab's sub menu items and then click the Export button and take all the defaults. This will create a TrustedRootCert.der file in the user's home directory.

4 Set the server certificate to be used for an SSL connection.

4a Select the properties of the LDAP Server - linux object, and then click on the SSL/TLS Configuration tab.

4b Click on the browse button for Server Certificate and select the SSL CertificateDNS object.

4c Click Apply, and then click OK.

Creating a New User in eDirectory

1 Select the novell container object and then click the New User button on the toolbar.

2 Enter the Name and Surname fields and click the OK button and enter the NDS password.

3 Close ConsoleOne.

Adding a Trusted Root Certificate to a Sun Keystore

1. Create a certificates directory in your user's home directory.
   - **1a** From a command prompt window, execute the following command:
     ```
     java sun.security.tools.KeyTool -import -alias TrustedCert -file ~/
     TrustedRootCert.der -keystore~/certs/sslkey.keystore
     ```
2. Use the SetClrPwdApp application to set the clear text password on a user object and assign a different password than the user's NDS password.
3. Build the clear password utility by executing the `build.sh`
   `/opt/novell/ndk/`*`nmas ndk`*`/nmas_sample_code/login_method/mgmt/ldap/build.sh`
4. Run the utility by executing the `run.sh`
   `/opt/novell/ndk/`*`nmas ndk`*`/nmas_sample_code/login_method/mgmt/ldap/run.sh`

3.7 Testing Your NMAS Methods

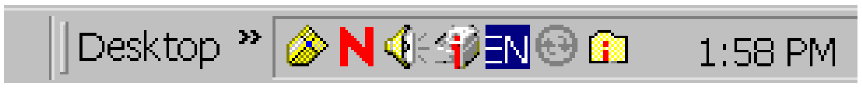
Once your NMAS methods are created, you can test them by following the procedures described in the following sections:

- Section 3.7.1, "Testing on Windows with the Novell Client," on page 40
- Section 3.7.2, "Testing the Linux LCM," on page 42
- Section 3.7.3, "Testing the Java LCM," on page 43

3.7.1 Testing on Windows with the Novell Client

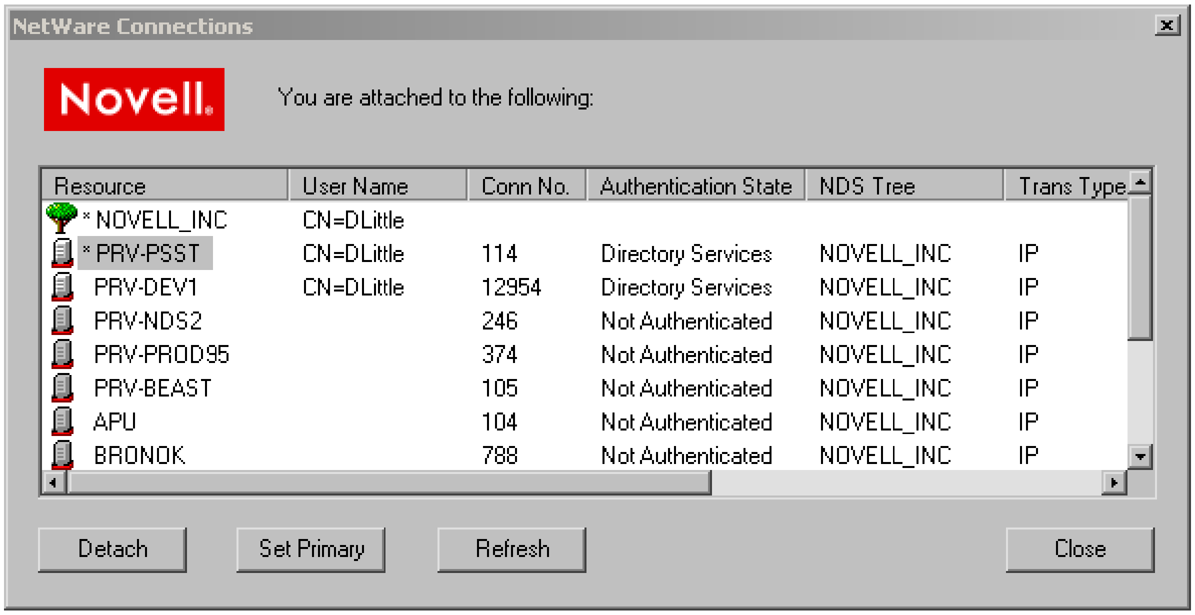
1. Right click on the N in the icon tray at the bottom of the screen.

Figure 3-3 *Windows Icon Tray*



2. Select NetWare Connections.

Figure 3-4 *NetWare Connections Tree*

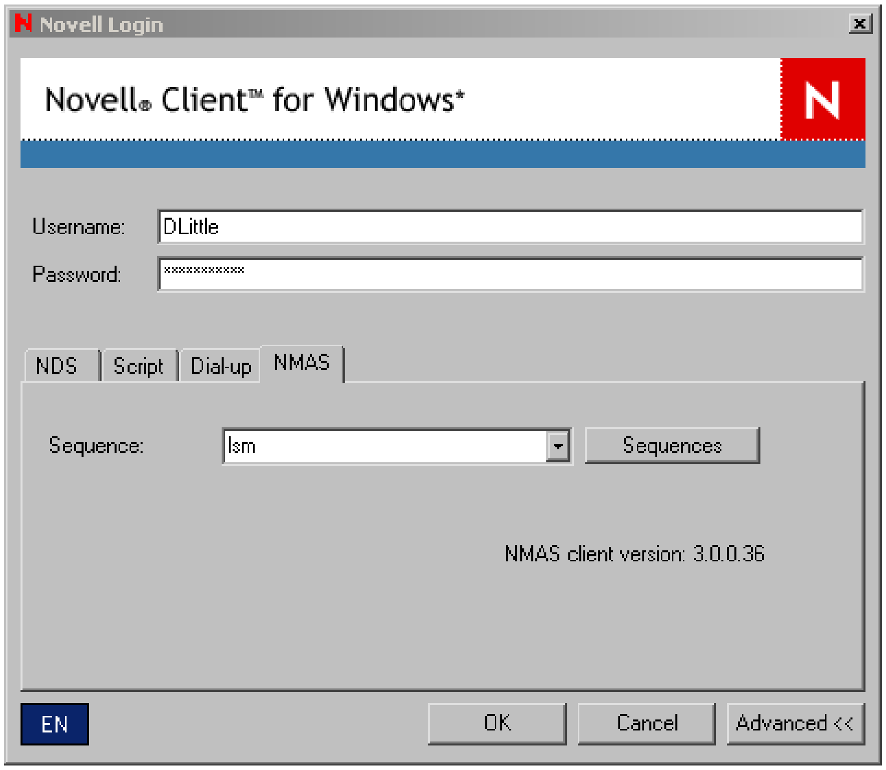


You are attached to the following:

Resource	User Name	Conn No.	Authentication State	NDS Tree	Trans Type
* NOVELL_INC	CN=DLittle				
* PRV-PSST	CN=DLittle	114	Directory Services	NOVELL_INC	IP
PRV-DEV1	CN=DLittle	12954	Directory Services	NOVELL_INC	IP
PRV-NDS2		246	Not Authenticated	NOVELL_INC	IP
PRV-PROD95		374	Not Authenticated	NOVELL_INC	IP
PRV-BEAST		105	Not Authenticated	NOVELL_INC	IP
APU		104	Not Authenticated	NOVELL_INC	IP
BRONOK		788	Not Authenticated	NOVELL_INC	IP

3 Right click on the N again in the Windows icon tray and select Netware Login.

Figure 3-5 *Novell Login GUI*



Username: DLittle

Password: ************

Sequence: lsm

NMAS client version: 3.0.0.36

4 Enter the username you assigned a clear text password to.

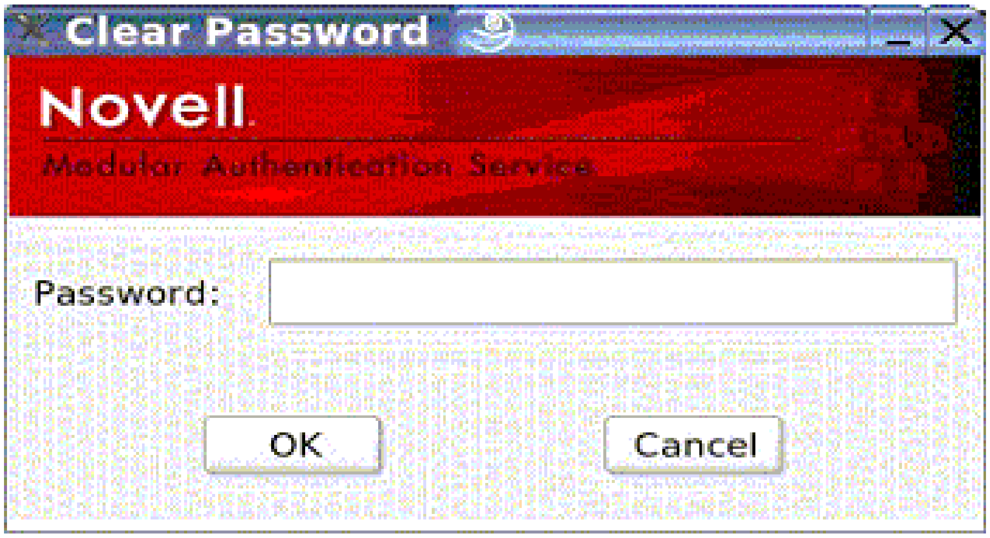
5 Click the Advance button and then select the NMAS tab.

6 Click on the Sequence browse button, then select lsm from the list, and then press OK to close the window.

7 Click on OK to login.

8 When prompted enter the clear text password you assigned to the user.

Figure 3-6 *NMAS Clear Password GUI*



3.7.2 Testing the Linux LCM

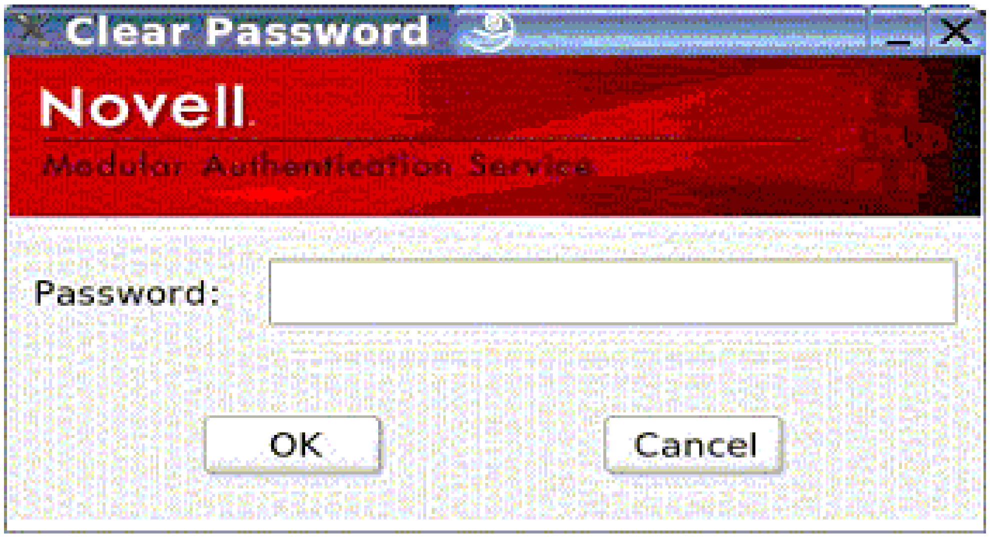
1 Build the sample saslbind application at `/opt/novell/ndk/`*`nmas ndk`*`/nmas_sample_code/nmas_client/` by simply entering `make` to execute the makefile.
`/opt/novell/ndk/`*`nmas ndk`*`/nmas_sample_code/nmas_client/make`

2 Execute the command `/opt/novell/ndk/`*`nmas ndk`*`/nmas_sample_code/nmas_client/saslbind application`.

Example: jdoe@linux:/opt/novell/ndk/*nmas_2005.02.24*/nmas_sample_code/nmas_client > ./saslbind localhost 389 cn=jdoe,o=*novell* lsmcpwd

3 At the pop-up window, enter the clear text password, and then click OK.

Figure 3-7 *NMAS Clear Password GUI*



4 In the terminal window, you should see that the login was successful.

Example: NMAS SASL Bind:

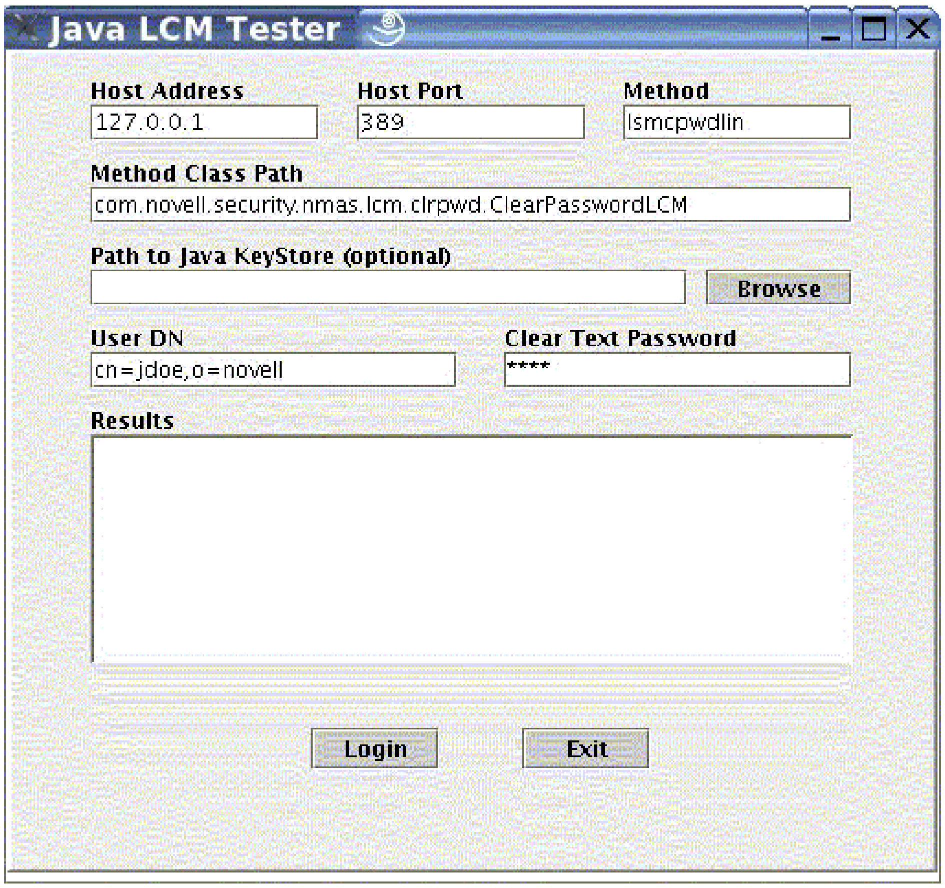
ldap host : localhost
ldap port : 389
ldap bind DN : cn=*jdoe*,o=*novell*
nmas sequence : lsmcpwd
password :

```
NMAS SASL Bind was successful
```

3.7.3 Testing the Java LCM

1. Build the Java LCM test application by executing
   `/opt/novell/ndk/`*`nmas ndk`*`/nmas_sample_code/nmas_java_client/ldap_login/build.sh`
2. Test the Java LCM by executing the LCMTestApp
   `/opt/novell/ndk/`*`nmas ndk`*`/nmas_sample_code/nmas_java_client/ldap_login/run.sh`

Figure 3-8 *Java LCM Tester GUI*



Host Address: 127.0.0.1
Host Port: 389
Method: lsmcpwdlin
Method Class Path: com.novell.security.nmas.lcm.clrpwd.ClearPasswordLCM
Path to Java KeyStore (optional):
User DN: cn=jdoe,o=novell
Clear Text Password: ****

4 Multiple Authentication Framework Functions

The Multiple Authentication Framework (MAF) functions allow you to develop a Login Server Module (LSM) and a Login Client Module (LCM). LSMs are written in C, while LSMs can be written in both the C and Java development languages. The following sections describe how these functions are used to develop LSMs and LCMs.

4.1 MAF Programming Model

To access NMAS™ and communicate with each other, the LCM and LSM follow a client/server model, where the NMAS login method invokes and governs the functions. These functions cannot be called directly from other applications.

Data is exchanged between the LCM and the LSM through the MAF_Write and MAF_Read (or the MAF_WriteRead), or the MAF_XWrite and MAF_XRead (or MAF_XWriteRead). The MAF_XWrite and MAF_XRead functions encrypt the data before it is transported and decrypt it after it is transported.

The MAF_GetAttribute function is available for the method to obtain data previously provided to Novell Modular Authentication Services (NMAS), such as username, password, requested clearance, and additional authentication information.

The modules must both execute the MAF_End function to provide NMAS with the status of this login method. A zero status specifies success and a non-zero status specifies failure. The module function return value is always zero (success) unless there is a fatal system error.

Examples of fatal system errors include a failure in the communication between the LCM and LSM (for example, MAF_Read or MAF_Write return an error) or failure to allocate memory. If an error (non-zero) value is returned from the LCM or LSM, NMAS is terminated and the login session fails.

Sample Login Method: The following code shows the basic transfer of authentication data from the LCM to the LSM:

Login Client Module (LCM)	Login Server Module (LSM)
`LCM00000001 (MAF_HANDLE mh)`	`LSM00000001 (MAF_HANDLE mh)`
`{`	`{`
`MAF_Begin(mh);`	`MAF_Begin(mh);`
Obtain login information from NMAS.	
`MAF _GetAttribute (mh,....`	
LCM processes data to send to LSM.	`---> MAF_Read(mh,&rlen,rbuf);`
	LSM processes Read
`MAF_WriteRead(mh,wlen,wbuf,&rlen,rbuf )`	`<--- MAF_Write(mh,wlen,wbuf);`
LSM processes Read	
WriteRead can be used either in the LCM or LSM so that multiple exchanges can be done.	WriteRead can be used either in the LCM or LSM so that multiple exchanges can be done.
`MAF_End(mh,status,0,0);`	`MAF_End(mh,status,0,0);`
`return(0); // Unless FATAL error`	`return(0); // Unless FATAL error`
`}`	`}`

4.2 NMAS Attribute IDs

The following table describes the NMAS Attribute Identifiers and whether or not they can be used for MAF_GetAttribute and MAF_PutAttribute (page 72) calls, and whether or not they require a non-null `tag` parameter.

Name	C Value	Requires a non-null `tag` parameter	MAF_ Put Attribute	NMAS_ Put Attribute	Description
NMAS_AID_USERNAME	1			X	Unicode* string that can contain the typeless, dot-delimited representation of the user distinguished name.
NMAS_AID_TREENAME	2			X	Unicode string that can contain the tree name.
NMAS_AID_DS_CONTEXT	3			X	Deprecated.
NMAS_AID_REQUESTED_ CLEARANCE	4			X	Unicode string that can contain a clearance entered by the user.
NMAS_AID_REQUESTED_READ_ CLEARANCE (Obsolete; replaced with NMAS_AID_REQUESTED_ CLEARANCE.)	4			X	Unicode string that can contain a clearance entered by the user.
NMAS_AID_REQUESTED_WRITE_ CLEARANCE	5				Deprecated.
NMAS_AID_PASSWORD	6		X	X	Unicode string that can contain a password entered by the user.
NMAS_AID_PIN	7		X	X	Unicode string that can contain a PIN entered by the user; only used internally by NMAS.
NMAS_AID_NAME_SPACE	8				Unicode string. This attribute is currently not used by NMAS.
NMAS_AID_ROLE	9				Unicode string. This attribute is currently not used by NMAS.
NMAS_AID_CLEARANCE_FLAG	10				Integer value. This attribute is currently not used by NMAS.
NMAS_AID_REQUESTED_ METHODS	11			X	Unicode string which contains the login sequence requested by the user.

Name	C Value	Requires a non-null `tag` parameter	MAF_ Put Attribute	NMAS_ Put Attribute	Description
NMAS_AID_MAF_CONTEXT	12				This attribute is for NMAS internal use only.
NMAS_AID_SERVER_MAF_ HANDLE	13				This attribute is for NMAS internal use only.
NMAS_AID_CLIENT_MAF_ HANDLE	14				This attribute is for NMAS internal use only.
NMAS_AID_SESSION_CONTEXT	15				This attribute is for NMAS internal use only,
NMAS_AID_CONNECTION	16			X	This attribute is for NMAS internal use only.
NMAS_AID_LDAP_CONTEXT	17				This attribute is for NMAS internal use only.
NMAS_AID_LM_CONFIG_DATA	18	X	X		Method-specific data stored in the login configuration store attribute of the login method object. Only available to LSMs.
NMAS_AID_LM_SECRET_DATA	19	X	X		Method-specific data stored in the login secret store attribute of the login method object. Only available to LSMs.
NMAS_AID_LD_CONFIG_DATA	20	X	X		Currently not supported. Intended to be method-specific data stored in the login configuration store attribute of the login device object. Only available to LSMs.
NMAS_AID_LD_SECRET_DATA	21	X	X		Currently not supported. Intended to be method-specific data stored in the login secret store attribute of the login device object. Only available to LSMs.
NMAS_AID_USER_CONFIG_DATA	22	X	X		Method-specific data stored in the login configuration store attribute of the user object. Only available to LSMs.
NMAS_AID_USER_SECRET_DATA	23	X	X		Method-specific data stored in the login secret store attribute of the user object. Only available to LSMs.

Name	C Value	Requires a non-null `tag` parameter	MAF_ Put Attribute	NMAS_ Put Attribute	Description
NMAS_AID_USER_GLOBAL_ CONFIG_DATA	24	X	X		Global data (data available to all login methods) stored in the login configuration store attribute of the user object. Only available to LSMs.
NMAS_AID_USER_GLOBAL_ SECRET_DATA	25	X	X		Global data (available to all login methods) stored in the login secret store attribute of the user object. Only available to LSMs.
NMAS_AID_NETWORK_ADDRESS	26			X	Binary string that can contain the network address (for example, IP or IPX) of the client. Only available to LSMs.
NMAS_AID_LCM_CALLBACK	27			X	Callback routine used by a proxy LCM to send data back to the proxy.
NMAS_AID_GA_INFO	28				Binary data that represents the clearance that is assigned to the session when login has completed successfully. Only available to LSMs.
NMAS_AID_DISCONNECTED_ FLAG	29				Indicates whether NMAS is running in disconnected mode. Only available to LCMs. Not currently supported.
NMAS_AID_LM_DN	30				Unicode string that contains the typeless, dot-delimited distinguished name of the login method object associated with the current login method. Only available to LSMs.
NMAS_AID_LM_GLOBAL_ CONFIG_DATA	31		X		Global data (available to all login methods) stored in the login configuration store attribute of the login method object. Only available to LSMs.

Name	C Value	Requires a non-null `tag` parameter	MAF_ Put Attribute	NMAS_ Put Attribute	Description
NMAS_AID_LM_GLOBAL_SECRET_DATA	32		X		Global data (available to all login methods) stored in the login secret store attribute of the login method object. Only available to LSMs.
NMAS_AID_GET_PASSWORD_POLICY_DN	33				Unicode string that contains the typeless, dot-delimited distinguished name of the password policy object associated with the user object. Only available to LSMs.
NMAS_AID_PROMPT_FOR_PASSWORD	34				Indicates to the Windows LCM that a password was not provided by calling the application and that the LCM is expected to prompt for the password if necessary. Only available to Windows LCMs.
NMAS_AID_OPTIONS	35				Options in the MAF Handle, only for MAF_GetAttribute (page 59). Only available to LCMs.
NMAS_AID_FEATURES	36				Not currently used. Only available to LCMs. MAF_GetAttribute.
NMAS_AID_PWD_WARNING	37		X		eDirectory™ error code returned by an NDS login method when the password is expired and is using grace logins. Only available to LCMs. (See MAF_GetAttribute (page 59).)
NMAS_AID_VERIFY_ONLY	38				Indicates that the user secrets are only verified. All login restrictions except for intruder lockout are ignored when this AID is set to a non-zero value. Only available to LSMs. (See MAF_GetAttribute (page 59).)

4.2.1 NMAS AID Options

The following values of MAF_GetAttribute (page 59) might be returned when the AID is NMAS_AID_OPTIONS:

Define Name	Value	Description
NMAS_OPT_NO_PROMPT_FOR_PWD	0x00000020	Does not prompt for a method password.
NMAS_OPT_LOGIN_ONLY	0x00000040	Does a login only, without allowing a password change.
NMAS_OPT_NO_METHOD_UI	0x00000080	The method should not display any user interface.
NMAS_OPT_VERIFY	0x00000100	Verifies the password without logging in.

4.3 MAF C Functions

4.3.1 MAF Attribute Functions

These functions might be used by LCMs and LSMs to retrieve login session information and user information.

- MAF_GetAttribute (page 59)
- MAF_PutAttribute (page 72)

4.3.2 MAF Login State Functions

These functions must be called by LCMs and LSMs to record state information with NMAS:

- MAF_Begin (page 56)
- MAF_End (page 57)

4.3.3 MAF Memory Functions

These functions can be used by LCMs and LSMs to allocate and free memory from the heap:

- MAF_MemFree (page 68)
- MAF_MemMalloc (page 69)
- MAF_MemRealloc (page 70)

4.3.4 MAF Password Functions

These functions can be used by LCMs and LSMs to send information between the LSM and the LCM:

- MAF_AllowPasswordSet (page 55)
- MAF_GetPassword (page 61)
- MAF_PolicyCheck (page 71)
- MAF_SetPassword (page 75)

4.3.5 MAF Trace Message Functions

These functions enable the LSM to write trace messages to the eDirectory trace log:

- MAF_Trace (page 76)
- MAF_TraceEnabled (page 77)
- MAF_TraceOnError (page 78)

4.3.6 MAF Transport Functions

These functions can be used by LCMs and LSMs to send information between the LSM and the LCM:

- MAF_Read (page 73)
- MAF_Write (page 79)
- MAF_WriteRead (page 80)
- MAF_XRead (page 82)
- MAF_XWrite (page 84)
- MAF_XWriteRead (page 85)

4.3.7 MAF Event Log Function

This function allows login methods to add audit log events to the NMAS audit log:

MAF_LogEvent (page 65)

4.3.8 MAF Data Store Functions***

The MAF Data Store Functions (MAFDS) functions are intended to replace the NWDS functions and new login methods should use the MAFDS functions. We created the MAFDS API to replace NWDS functions that are not available on the UNIX platforms.

Developers should use the MAFDS API if you:

- Have used NWDS functions in your previous methods.
- Need to read data from eDirectory that's not accessible from calling MAF_GetAttribute (page 59). MAF_GetAttribute is sufficient for most developers.

The following functions may be used by LSMs to read objects and attributes from eDirectory:

- MAFDS_CreateContext (page 88)

- MAFDS_FreeContext (page 90)
- MAFDS_GetValueData (page 96)
- MAFDS_InsertModValue (page 97)
- MAFDS_ListContainerEntries (page 99)
- MAFDS_ModifyEntry (page 100)
- MAFDS_ReadAttributeValues (page 102)
- MAFDS_VALUE_DATA (page 106)

NOTE: In NMAS 3.1, these functions should be used to create all new NMAS methods instead of previous NWDS functions.

MAFDS Operation Constants

The following values are options for MAFDS_InsertModValue (page 97):

Table 4-1 *Values for the "operation" parameter of MAFDS_InsertModValue.*

Value	Description
MAFDS_MODOP_ADD_ATTR	Adds a value to an attribute.
MAFDS_MODOP_MODIFY	Changes a value for a single-valued attribute, or adds a value to a multi-valued attribute.
MAFDS_MODOP_DEL_VALUE	Removes a value from an attribute.
MAFDS_MODOP_DEL_ATTR	Removes an attribute and all of its values.

MAFDS Syntax Constants

Table 4-2 *Syntax constants for the MAFDS functions.*

Value	Description
MAFDS_SYN_DISTNAME	Specifies the Distinguished Name.
MAFDS_SYN_CE_STRING	Specifies a case-exact string.
MAFDS_SYN_CI_STRING	Specifies a case-ignore string.
MAFDS_SYN_BOOLEAN	Specifies a Boolean value.
MAFDS_SYN_INTEGER	Specifies an integer.
MAFDS_SYN_OCTET_STRING	Specifies a string containing binary data.
MAFDS_SYN_TIME	Specifies time in seconds (`time_t`).

4.4 MAF Java Functions

The NMAS MAF Java library provides methods to develop a Login Server Module (LSM) and a Login Client Module (LCM), as described in Section 4.1, "MAF Programming Model," on page 46. The common Java interface transforms the function-based C client API into an object-oriented Java client API. Because the library is Java-based, it is portable across multiple platforms. This extensible, modular design enables new library features to be added without impacting applications that use its functions.

4.4.1 Library Architecture

The NMAS library provides a common, object-oriented interface to NMAS. Additional implementations can be added to the library without impacting the Java applications. The Java interfaces uses the following classes:

- **MAF—** Defines the Multiple Authentication Framework (MAF) to create a login server. For specific implementation of this class, see the MAF.java class (../api/index.html).
- **ClearPasswordLCM—** Creates the clear text password method for the login client module (LCM). For specific implementation of this class, see the ClearPasswordLCM class (../api/index.html).

4.4.2 NMAS Java LCM on Linux

The MAF class makes up most of the API. The MAF class manages authentication, while most other classes deal with functionality involving available Java NMAS login methods.

Currently there is no support for NMAS Client methods on Linux. However, it is possible to use a Java LCM on Linux to achieve authentication from a Linux client. Also, a Java LCM makes it possible to authenticate from a servlet or portlet.

To configure and test the example Clear Text Password method in the NMAS NDK on Linux using a Java LCM and a test application, follow the procedures provided in NMAS Java LCM on Linux (http://developer.novell.com/wiki/index.php/Java_LCM_for_Linux). To see clear text sample method, go to the Java LCM sample (http://developer.novell.com/ndk/doc/samplecode/nmas_sample/index.htm).

You should also be familiar with the following NDK sections:

Section 3.4, "Installing the Clear Text Password Method," on page 34
Section 3.5, "Building an NMAS LCM," on page 34
Section 3.6, "Enrolling a User with the Clear Text Password," on page 36

4.5 MAF C Function Listing

The following section contains the following Multiple Authentication Framework (MAF) C functions, which enable LSMs to retrieve and manage the user's password:

MAF_AllowPasswordSet

Allows an LSM to determine if the user is allowed to set the password for the object that is associated with this login session. This function is available only in NMAS 2.2 or later.

Syntax

```
#include <maf.h>

nint32   MAF_AllowPasswordSet
(
   MAF_Handle   mh
);
```

Parameters

mh

(IN) Specifies the MAF handle.

Return Codes

Returns 0 if the user is allowed to set the password or a nonzero NMAS Error Codes (page 173) if the user is not allowed to set the password.

Remarks

MAF_AllowPasswordSet provides information to the LSM if the user is allowed to set his or her password.

See Also

MAF_GetPassword (page 61)
MAF_SetPassword (page 75)

MAF_Begin

Directs the NMAS server to send the DO operation to the NMAS client. The NMAS Client invokes the LCM when it receives a “DO” operation. It is called after the LSM/LCM starts execution. This function lets NMAS know that the module is fully initialized and ready to begin communication.

Syntax

```
#include <maf.h>

nint32    MAF_Begin
(
   MAF_Handle   mh
);
```

Parameters

mh

Specifies the MAF handle as supplied to the LSM/LCM by NMAS.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

All errors indicate that the login session is unusable. If errors occur, the method should free all the resources it holds and return a non-zero value.

Remarks

This function directs the NMAS Server to send the DO operation to the NMAS client to invoke the LCM only when the MAF_Begin is called by the LSM.

See Also

MAF_End (page 57)

MAF_End

Lets NMAS know whether the method operation was a success (zero) or failure (non-zero). This information is used by NMAS to determine the next required login steps.

Syntax

```
#include <maf.h>

nint32   MAF_End
(
   MAF_Handle   mh,
   nint32       status,
   nint32       len,
   void        *seedDatap
);
```

Parameters

mh

Specifies the MAF handle as supplied to the LCM/LSM.

status

0 if LCM/LSM completed successfully; non-zero if LCM/LSM failed in its login operations.

len

Reserve code is 0.

seedDatap

Reserve code is null.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

Remarks

This function is the last MAF function called before the LCM/LSM returns.

The method must still execute a return after calling this function. The return value normally is 0 (SUCCESS) unless a fatal system error occurs. If an error (non-zero) value is returned, NMAS is terminated and the login session fails.

See Also

MAF_Begin (page 56)

MAF_Free (obsolete 3/1/2006)

Zeroed out the memory before it is returned to the heap but is now replaced by MAF_MemFree (page 68).

Syntax

```
#include <maf.h>

nint32   MAF_Free
(
   void      *p,
);
```

Parameters

p

Points to a buffer allocated using MAF_Malloc.

See Also

MAF_Malloc (obsolete 3/1/2006) (page 67)

MAF_GetAttribute

Allows an LCM/LSM to get the value of an attribute that is associated with this login session.

Syntax

```
#include <maf.h>

nint32   MAF_GetAttribute
(
   MAF_Handle   mh,
   nint32       aid,
   unicode     *tag,
   pnint32     *aLenp,
   void        *aValuep
);
```

Parameters

mh

(IN) Specifies the MAF handle.

aid

(IN) The NMAS for the attribute (see NMAS Attribute IDs (page 47)).

tag

(IN) The `tag` parameter is a null-terminated Unicode string that matches the tag associated with the data that has been written by *MAF_PutAttribute*, or the management functions *NMAS_PutLoginConfig* and *NMAS_PutLoginSecret*. This parameter is required for some AIDs. It should be null for other AIDs. For more information about the tag parameter, see NMAS Attribute IDs (page 47).

aLenp

(IN/OUT) *IN:* Length of buffer; *OUT:* Length of the attribute value in `aValuep`.

aValuep

(OUT) Points to a memory area where this function places the request value. To see what values might be returned when the aid is NMAS_AID_OPTIONS, see NMAS AID Options (page 51).

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

Remarks

The length of the attribute is returned to `aLenp` by first calling *MAF_GetAttribute* with `aLenp` set to zero and null for `aValuep`. In this case, an error *NMAS_E_BUFFER_OVERFLOW* (-1633) might be returned.

Because the Simple Password is used by several different methods (Simple Password method, CIFS methods, and AFP methods), it is designed to be user global configuration data. User global configuration data is data that any login method LSM can retrieve by calling MAF_GetAttribute (page 59) using the AID of NMAS_AID_USER_GLOBAL_CONFIG_DATA. You store user global configuration data in the login configuration data by passing in `0x00` as the fourth parameter to NMAS_PutLoginConfig (page 115).

See Also

MAF_PutAttribute (page 72)
NMAS_PutLoginConfig (page 115)
NMAS_PutLoginSecret (page 117)

MAF_GetPassword

Returns the Universal Password password and password information of the user associated with the current login session (LSM only). This function is available only in NMAS 2.2 or later.

Syntax

```
#include <maf.h>

int MAF_GetPassword
(
   MAF_Handle   mh,
   size_t      *passwordLen,
   unicode     *password,
   nuint32     *infoFlags,
   nint32      *remainingLifetime,
   nint32      *remainingGraceLogins
);
```

Parameters

`mh`

(IN) Specifies the MAF handle.

`passwordLen`

(IN/OUT) *IN*: Specifies the length in Unicode characters of the password buffer and return. *OUT*: Specifies the length in Unicode characters of the password, including the null termination.

`password`

(OUT) Points to the buffer to which the null-terminated Unicode password is copied. This optional parameter can be null.

`infoFlags`

(OUT) Points to additional information about the current password; for example, indicating the password has expired. This optional parameter can be null.

Valid Flag Defines	Value	Description
SPM_EXPIRED	0x1	Indicates the number of seconds since the password expired.
SPM_GRACE_LOGIN_LIMIT	0x2	Indicates the number of additional times the user can log in without changing the password.
SPM_SPWD_MIGRATED	0x4	Indicates that the returned password is the simple password and that the password was set using the simple password.

`remainingLifetime`

(OUT) If the output SPM_EXPIRED bit is not set in the `infoFlags`, contains the number of seconds that remain before the password expires. If the SPM_EXPIRED bit is set in the `infoFlags`, contains the number of seconds since the password expired. The value of SPM_NO_PASSWORD_EXPIRATION (0XFFFFFFFF) indicates that there is no password expiration time. If set to null, no password expiration is returned.

`remainingGraceLogins`

(OUT) If the SPM_EXPIRED bit is set in `infoFlags`, indicates the number of additional times the user can log in without changing the password. The value of SPM_UNLIMITED_GRACE_LOGIN (0xFF) indicates there is no limit to the number of grace logins. If set to null, no grace login information is returned.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

Remarks

If no NMAS password has been set for the user and the user has enabled a simple password, the Universal Password is set to the value of the simple password. If this is the case, the SPM_SPWD_MIGRATED flag in the *infoFlags* is set. If the buffer is not large enough, an error NMAS_E_BUFFER_OVERFLOW (-1633) might be returned. PasswordLen then contains the required buffer size (in unicode characters).

See Also

MAF_AllowPasswordSet (page 55)
MAF_Free (obsolete 3/1/2006) (page 58)
MAF_SetPassword (page 75)

MAF_GetPasswordEx

Returns the Universal Password password and password information of the user associated with the current login session (LSM only). This function is available only in NMAS 2.3 or later.

Syntax

```
#include <maf.h>

int MAF_GetPasswordEx
(
   MAF_Handle   mh,
   nint32      *getFlags,
   size_t      *passwordLen,
   unicode     *password,
   nuint32     *infoFlags,
   nint32      *remainingLifetime,
   nint32      *remainingGraceLogins
);
```

Parameters

mh

(IN) Specifies the MAF handle.

getFlags

(IN) Flags that effect the behavior of this function are defined in the following table:

Valid Flag Defines	Value	Description
SPM_SYNC_REQUEST	0x1	Does not enforce password expiration and grace login.
SPM_INFO_REQUEST	0x2	Returns only password information (length, creation time, expiration and grace logins).
SPM_DONT_MIGRATE_SPWD	0x4	Does not copy simple password to password.
SPM_IGNORE_NDS	0x8	Does not check if the eDirectory password is newer than password.
SPM_IGNORE_EXPIRATION	0x10	Does not read or check password expiration and grace login.

passwordLen

(IN/OUT) *IN*: Specifies the length in Unicode characters of the password buffer and return. *OUT*: Specifies the length in Unicode characters of the password, including the null termination.

password

(OUT) Points to the buffer to which the null-terminated Unicode password is copied. This optional parameter can be null.

`infoFlags`

(OUT) Points to additional information about the current password; for example, indicating the password has expired. This optional parameter can be null.

Valid Flag Defines	Value	Description
SPM_EXPIRED	0x1	Indicates the number of seconds since the password expired.
SPM_GRACE_LOGIN_LIMIT	0x2	Indicates the number of additional times the user can log in without changing the password.
SPM_SPWD_MIGRATED	0x4	Indicates that the returned password is the simple password and that the password was set using the simple password.

`remainingLifetime`

(OUT) If the output SPM_EXPIRED bit is not set in the `infoFlags`, contains the number of seconds that remain before the password expires. If the SPM_EXPIRED bit is set in the `infoFlags`, contains the number of seconds since the password expired. The value of SPM_NO_PASSWORD_EXPIRATION (0XFFFFFFFF) indicates that there is no password expiration time. If set to null, no password expiration is returned.

`remainingGraceLogins`

(OUT) If the SPM_EXPIRED bit is set in `infoFlags`, indicates the number of additional times the user can log in without changing the password. The value of SPM_UNLIMITED_GRACE_LOGIN (0xFF) indicates there is no limit to the number of grace logins. If set to null, no grace login information is returned.

Return Code

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

Remarks

If the SPM_DONT_MIGRATE_SPWD flag is not set on the in-out flags and no NMAS password has been set for the user and the user has a simple password, the Universal Password is set to the value of the simple password. If this is the case, the SPM_SPWD_MIGRATED flag in the `infoFlags` is set.

See Also

MAF_AllowPasswordSet (page 55)
MAF_Free (obsolete 3/1/2006) (page 58)
MAF_GetPassword (page 61)
MAF_SetPassword (page 75)

MAF_LogEvent

Allows login methods to add audit log events to the NMAS audit log. This function is only available to proxy LCMs and LSMs. This functions is available only in NMAS 3.0 or later.

Syntax

```
#include <maf.h>

nuint32 MAF_LogEvent
(
   MAF_Handle       mh,
   nuint32          logLevel,
   nuint32          status,
   unsigned char   *statusMsg,
   nuint32          methodEventID,
   unsigned char   *eventMsgFmt,
   [...]
);
```

Parameters

`mh`

(IN) Specifies the MAF handle.

`logLevel`

(IN) Specifies the importance of the log event. The possible values are as follows:

Define	Value
MAF_LOG_LEVEL_EMERGENCY	1
MAF_LOG_LEVEL_ALERT	2
MAF_LOG_LEVEL_CRITICAL	3
MAF_LOG_LEVEL_ERROR	4
MAF_LOG_LEVEL_WARNING	5
MAF_LOG_LEVEL_NOTICE	6
MAF_LOG_LEVEL_INFO	7
MAF_LOG_LEVEL_DEBUG	8

`status`

(IN) Specifies if the requested operation was success (zero value) or failed (non-zero value).

`statusMsg`

(IN) An message that describes the status of the requested operation.

`methodEventID`

(IN) An identifier that specifies the method specific audit log event identifier.

`eventMsgFmt`

(IN) Text that will be included in the log event. The text may optionally can contain format tags that will be replaced by the values specified in the remaining arguments. The format tags are the standard sprintf format tags.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

MAF_Malloc (obsolete 3/1/2006)

Allocated memory for the LSM and LCM but is now replaced by MAF_MemMalloc (page 69) and MAF_MemRealloc (page 70).

Syntax

```
#include <maf.h>

nint32   MAF_Malloc
(
   size_t      s,
);
```

Parameters

s

Size of the buffer to be allocated.

See Also

MAF_Free (obsolete 3/1/2006) (page 58)

MAF_MemFree

Frees a buffer allocated with MAF_MemMalloc (page 69).

Syntax

C

```
#include <maf.h>

NMASAPI void MAF_MemFree
(
   MAF_Handle    mh,
   void         *p
);
```

Parameters

mh

(IN) The MAF_Handle passed to your method's start routine.

p

(IN) A pointer to a buffer allocated using MAF_MemMalloc (page 69) or MAF_MemRealloc (page 70).

MAF_MemMalloc

Allocates memory to be used by a login method. Memory should be released using the MAF_MemFree (page 68) method.

Syntax

C

```
#include <maf.h>

NMASAPI void *MAF_MemMalloc
(
   MAF_Handle   mh,
   size_t       s
);
```

Parameters

mh

(IN) The MAF_Handle passed to your method's start routine.

s

(IN) The size of the buffer to be allocated.

Return Values

Returns a pointer to the newly allocated buffer on success, NULL on failure.

Remarks

This call should be used instead of MAF_Malloc (obsolete 3/1/2006) (page 67).

MAF_MemRealloc

Reallocates a buffer previously allocated with MAF_MemMalloc (page 69).

Syntax

C

```
#include <maf.h>

NMASAPI void *MAF_MemRealloc
(
   MAF_Handle   mh,
   void        *p,
   size_t       s
);
```

Parameters

mh

(IN) The MAF_Handle passed to your method's start routine

p

(IN) A pointer to a buffer allocated using MAF_MemMalloc (page 69), or NULL to allocate a new buffer.

s

(IN) The size of the buffer to be allocated.

Return Values

Returns a pointer to the newly allocated buffer on success, NULL on failure.

Remarks

Use this call instead of MAF_Realloc.

MAF_PolicyCheck

Performs a password policy check using the password policy that is effective for the user (LSM only). This function is available only in NMAS 2.2 or later.

Syntax

```
#include <maf.h>

nint32   MAF_PolicyCheck
(
   MAF_Handle   mh,
   size_t       passwordLen,
   unicode     *password
);
```

Parameters

mh

(IN) Specifies the MAF handle.

passwordLen

(IN) The length in Unicode characters, including null termination, of the password.

password

(IN) Points to a null-terminated Unicode string that is checked against the current password format policies. If null, the user's currently active password is checked against the current policy.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

Remarks

If `passwordLen` is 0 and `password` is null, the user's current password is checked against the established login policy.

See Also

MAF_GetPassword (page 61)

MAF_PutAttribute

Allows an LCM/LSM to put the value of an attribute that is associated with this login session.

Syntax

```
#include <maf.h>

 nint32   MAF_PutAttribute
(
     MAF_Handle   mh,
     nint32       aid,
     unicode     *tag,
     nint32       aLen,
     void        *aValuep
);
```

Parameters

mh

Specifies the MAF handle.

aid

(IN) The NMAS for the attribute. (See Section 4.2, "NMAS Attribute IDs," on page 47.)

tag

(IN) The `tag` parameter is a null-terminated UNICODE string that matches the tag associated with the data. This parameter is required for some AIDs. For other `AID` parameters, the tag should be null. For more information about the tag parameter, see Section 4.2, "NMAS Attribute IDs," on page 47.

aLen

Length of the attribute value in `aValuep`.

aValuep

Points to a memory area where this function places the value to be stored.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

Remarks

If the parameters `aLen` and `aValuep` are 0 and null respectively, then both the attribute associated with the current method and the tag are removed.

See Also

MAF_GetAttribute (page 59)
NMAS_PutLoginConfig (page 115)
NMAS_PutLoginSecret (page 117)
NMAS_GetLoginConfig (page 111)

MAF_Read

The parameter `rLenp` returns the size of the request in bytes. `rBufp` is a local pointer to a buffer that has been pre-allocated by the caller. The format of the data that is copied into `rBufp` is defined by the method.

Syntax

```
#include <maf.h>

 nint32   MAF_Read
(
    MAF_Handle   mh,
    pnint32     *rLenp,
    void        *rBufp
);
```

Parameters

`mh`

Specifies the MAF handle.

`rLenp`

Returns the size in bytes of the data pointed to by *rBufp*.

`rBufp`

Points to a memory area where this function places the request data. This buffer must be large enough to receive the expected data.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

Returns *NMAS_E_NOT_SUPPORTED* if called while in disconnected mode.

All errors indicate that it is not possible for the LSM and LCM to communicate. If an error other than *NMAS_E_NOT_SUPPORTED* (as described above) is returned the module should return a non-zero value that causes the termination of the login session.

Remarks

The parameter *rLenp* returns the size of the request in bytes, while *rBufp* is a local pointer to a buffer that has been pre-allocated by the method that the data is copied to. The actual format of *rBufp* data is method-defined.

The method must always provide a write to each `MAF_Read` unless a MAF error occurs. The `MAF_Read` function is invoked by the LSM to read data from the LCM and is invoked by the LCM to read data from the LSM.

`MAF_Read` blocks until there is data to read or until it times out (after 3 minutes).

See Also

MAF_WriteRead (page 80)
MAF_Write (page 79)

MAF_SetPassword

Sets the password of the user associated with the current login session to the specified password (LSM only). This function is available only in NMAS 2.2 or later.

Syntax

```
#include <maf.h>

int MAF_SetPassword
(
   MAF_Handle   mh,
   size_t       passwordLen,
   unicode     *password,
   nuint32      setFlags
);
```

Parameters

mh

(IN) The MAF handle.

passwordLen

(IN) The length in Unicode characters of the candidate password, including the null termination.

password

(IN) Points to a null-terminated Unicode string that is the candidate password.

setFlags

(IN) Provides additional information from the agent to SPM. For example, the agent can use this flag to specify that the user or someone else set the password.

Valid Flag Defines	Value	Description
SPM_ADMIN_REQUEST	0x1	Indicates that the password set request was not from the user.
SPM_IGNORE_POLICY	0x8	Indicates that the password policy is not to be enforced when setting the password.
SPM_EXPIRE_NOW	0x10	Indicates that the password is marked as expired.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

MAF_AllowPasswordSet (page 55)
MAF_GetPassword (page 61)

MAF_Trace

Allows trace messages to be displayed in Directory Services Trace (DSTrace) only in LSMs and Proxy LCMs. This function is available only in NMAS 2.2 or later.

Syntax

```
#include <maf.h>

nint32   MAF_Trace
(
   MAF_Handle   mh,
   char        *message
);
```

Parameters

mh

(IN) Specifies the MAF handle.

message

Points to a null-terminated character string that contains a trace message.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

MAF_TraceEnabled (page 77)
MAF_TraceOnError (page 78)

MAF_TraceEnabled

Allows LSM and Proxy LCM to determine if NMAS trace is enabled. This function is available only in NMAS 2.2 or later.

Syntax

```
#include <maf.h>

nint32   MAF_TraceEnabled
(
   MAF_Handle   mh,
);
```

Parameters

`mh`

(IN) Specifies the MAF handle.

Return Codes

Returns non-zero if NMAS_Trace is enabled, otherwise returns 0.

See Also

MAF_Trace (page 76)
MAF_TraceOnError (page 78)

MAF_TraceOnError

Allows trace messages to be displayed in Directory Services Trace (DSTrace) in LSMS and Proxy LCMs. The message is displayed only if the input error is non-zero. This function is available only in NMAS 2.2 or later

Syntax

```
#include <maf.h>

nint32   MAF_TraceOnError
(
   MAF_Handle   mh,
   int          err,
   char        *message
);
```

Parameters

mh

(IN) Specifies the MAF handle.

err

(IN) If a non-zero message is displayed, an error is incorporated into the message.

message

(IN) Points to a null-terminated ASCII character string that contains a trace message.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

MAF_Trace (page 76)
MAF_TraceEnabled (page 77)

MAF_Write

Allows the LCM/LSM to write data to the LSM/LCM. This function sends *wlen* bytes of data to its counterpart.

Syntax

```
#include <maf.h>

 nint32    MAF_Write
(
    MAF_Handle   mh,
    nint32       wlen,
    void        *wBufp
);
```

Parameters

mh

Specifies the MAF handle.

wlen

The size in bytes of the data pointed to by *wBufp*.

wBufp

Pointer to the data to write.

Return Codes

***Returns 0 if successful or a non-zero NMAS Error Code if not successful.

Returns NMAS_E_NOT_SUPPORTED if called while in disconnected mode.

All errors indicate that it is not possible for the LSM and LCM to communicate. If an error other than *NMAS_E_NOT_SUPPORTED* (as described above) is returned the module should return a non-zero value that causes the termination of the login session.

Remarks

The actual format of the *wbufp* data is method defined.

You must always provide a read to each *MAF-Write* unless an MAF Transport error occurs.

See Also

MAF_Read (page 73)
MAF_WriteRead (page 80)

MAF_WriteRead

Writes and reads data in a single operation. This function is the same as MAF_Write (page 79) followed by a MAF_Read (page 73).

Syntax

```
#include <maf.h>

 nint32   MAF_WriteRead
(
    MAF_Handle   mh,
    nint32       wlen,
    void        *wBufp,
    pnint32     *rLenp,
    void        *rBufp
);
```

Parameters

mh

Specifies the MAF handle.

wLen

The size in bytes of the data pointed to by *wBufp*.

wBufp

Points to the data to write.

rLenp

Returns the size in bytes of the data pointed to by *rBufp*.

rBufp

Points to a memory area where this function places the requested data. Must be large enough to receive expected data.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

Returns *NMAS_E_NOT_SUPPORTED* if called while in disconnected mode.

All errors indicate that it is not possible for the LSM and LCM to communicate. If an error other than *NMAS_E_NOT_SUPPORTED* (as described above) is returned the module should return a non-zero value that causes the termination of the login session.

Remarks

MAF_WriteRead blocks until there is data to read or until it times out (after 3 minutes).

See Also

MAF_Read (page 73)
MAF_Write (page 79)

MAF_XWriteRead (page 85)

MAF_XRead

Allows the LCM/LSM to read encrypted data. This function receives data that was encrypted when sent by the LCM or LSM. The parameter *rLenp* returns the size of the request in bytes. *rBufp* is a local pointer to a buffer that has been pre-allocated by the caller. The format of the data that is copied into *rBufp* is defined by the method.

Syntax

```
#include <maf.h>

 nint32   MAF_XRead
(
    MAF_Handle    mh,
    pnint32      *rLenp,
    void         *rBufp
);
```

Parameters

mh

Specifies the MAF handle.

rLenp

Returns the size in bytes of the clear-text data pointed to by *rBufp*.

rBufp

Points to a memory area where this function places the requested data. Must be large enough to receive the expected data.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

Returns *NMAS_E_NOT_SUPPORTED* if called while in disconnected mode.

All errors indicate that it is not possible for the LSM and LCM to communicate. If an error other than *NMAS_E_NOT_SUPPORTED* (as described above) is returned, the module should return a non-zero value that causes the termination of the login session.

Remarks

The parameter *rLenp* returns the size of the request in bytes, while *rBufp* is a local pointer to a buffer that has been pre-allocated by the method. Clear-text data is copied to this buffer. The actual format of *rBufp* data is method-defined.

The method must always provide an encrypted write to each *MAF_XRead* unless a MAF error occurs.

See Also

MAF_WriteRead (page 80)
MAF_Write (page 79)
MAF_XWriteRead (page 85)

MAF_XWrite (page 84)

MAF_XWrite

Allows the LCM/LSM to write encrypted data to the LSM/LCM. The function sends *wlen* bytes of data to its counterpart that is encrypted for transit. The actual format of the *wBufp* data is method-defined.

Syntax

```
#include <maf.h>

 nint32   MAF_XWrite
(
    MAF_Handle   mh,
    nint32       wlen,
    void        *wBufp
);
```

Parameters

mh

Specifies the MAF handle.

wlen

The size in bytes of the clear-text data pointed to by *wBufp*.

wBufp

Points to the clear-text data to write.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

Returns *NMAS_E_NOT_SUPPORTED* if called while in disconnected mode.

All errors indicate that it is not possible for the LSM and LCM to communicate. If an error other than *NMAS_E_NOT_SUPPORTED* (as described above) is returned, the module should return a non-zero value that causes the termination of the login session.

Remarks

You must always provide an encrypted read to each *MAF_XWrite* unless a MAF Transport occurs.

See Also

MAF_Read (page 73)
MAF_WriteRead (page 80)
MAF_XRead (page 82)
MAF_XWriteRead (page 85)

MAF_XWriteRead

Writes and reads data in a single operation with both encrypted and decrypted data as appropriate. This function is the same as MAF_Write (page 79) followed by a MAF_XRead (page 82) .

Syntax

```
#include <maf.h>

 nint32   MAF_XWriteRead
(
    MAF_Handle   mh,
    nint32       wlen,
    void        *wBufp,
    pnint32     *rLenp,
    void        *rBufp
);
```

Parameters

mh

Specifies the MAF handle.

wLen

The size in bytes of the data pointed to by *wBufp*.

wBufp

Points to the clear-text data to write.

rLenp

Returns the size in bytes of the clear-text data pointed to by *rBufp*.

rBufp

Points to a memory area where this function places the requested data. Must be large enough to receive expected data.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

Returns *NMAS_E_NOT_SUPPORTED* if called while in disconnected mode.

All errors indicate that it is not possible for the LSM and LCM to communicate. If an error other than *NMAS_E_NOT_SUPPORTED* (as described above) is returned, the module should return a non-zero value that causes the termination of the login session.

Remarks

MAF_XWriteRead blocks until there is data to read or until it times out.

See Also

MAF_Read (page 73)
MAF_Write (page 79)

MAF_XRead (page 82)
MAF_XWrite (page 84)
MAF_WriteRead (page 80)

MAFDS_ATTRIBUTE

Contains information about a data store attribute.

Syntax

C

```
#include <mafds.h>

typedef struct MAFDS_ATTRIBUTE
{
   unicode   *attr;
   int        syntax;
};
```

Fields

attr

The name of the attribute.

syntax

The syntax attribute. (See "MAFDS Syntax Constants" on page 53 for a listing of possible values.)

MAFDS_CreateContext

Creates a new MAFDS context handle that can be used with other MAFDS routines.

Syntax

C

```
#include <mafds.h>

NMASAPI int MAFDS_CreateContext
(
   MAF_Handle     mh,
   MAFDS_CONTEXT *outContext
);
```

Parameters

`mh`

(IN) The MAF_Handle passed to your method's start routine.

`outContext`

(OUT) A pointer to a MAFDS_CONTEXT that will receive the new context handle.

Return Values

Zero on success, non-zero on failure.

Remarks

The context handle returned from this call will have rights to read and write objects stored in local replicas. The context handle will also have read-only access to the Security Container in the tree.

See Also

MAFDS_FreeContext (page 90)

MAFDS_FreeContainerEntries

Free a list of container entries returned from MAFDS_ListContainerEntries (page 99).

Syntax

C

```
#include <mafds.h>

NMASAPI int MAFDS_FreeContainerEntries
(
   MAFDS_CONTEXT   context
   unicode       **entries
);
```

Parameters

`context`

(IN) The context that was used to read the values.

`entries`

(IN) A array of unicode strings returned from MAFDS_ListContainerEntries (page 99).

See Also

MAFDS_ListContainerEntries (page 99)

MAFDS_FreeContext

Frees a MAFDS context.

Syntax

C

```
#include <mafds.h>

NMASAPI int MAFDS_FreeContext
(
   MAFDS_CONTEXT   context
);
```

Parameters

`context`

(IN) The context to be freed.

See Also

MAFDS_CreateContext (page 88)

MAFDS_FreeModValues

Frees a MAFDS_MODVALUE handle that was allocated using MAFDS_InsertModValue (page 97).

Syntax

C

```
#include <mafds.h>

NMASAPI int MAFDS_FreeModValues
(
    MAFDS_CONTEXT   context,
    MAFDS_MODVALUE *modValues
);
```

Parameters

`context`

(IN) A MAFDS context allocated using MAFDS_CreateContext (page 88).

`modValues`

(IN) A MAFDS_MODVALUE handle.

See Also

MAFDS_CreateContext (page 88)
MAFDS_InsertModValue (page 97)

MAFDS_FreeValues

Frees a MAFDS_VALUE handle that was allocated with either MAFDS_ReadAttributeValues (page 102) or MAFDS_ReadInheritedAttributeValues (page 104).

Syntax

C

```
#include <mafds.h>

NMASAPI int MAFDS_FreeValues
(
   MAFDS_CONTEXT   context
   MAFDS_VALUE     values
);
```

Parameters

context

(IN) The context that was used to read the values.

values

(IN) The MAFDS_VALUE handle to free.

See Also

MAFDS_ReadAttributeValues (page 102)
MAFDS_ReadInheritedAttributeValues (page 104)

MAFDS_FreeValueData

Free the contents of a value data structure returned from MAFDS_GetValueData (page 96).

Syntax

C

```
#include <mafds.h>

NMASAPI int MAFDS_FreeValueData
(
    MAFDS_CONTEXT      context,
    MAFDS_VALUE_DATA  *valueData
);
```

Parameters

`context`

(IN) The context that was used to read the values.

`valueData`

(IN) A pointer to a MAFDS_VALUE_DATA structure whose contents are to be freed.

Remarks

Use this call to free the contents of a MAFDS_VALUE_DATA structure that were retrieved using MAFDS_GetValueData (page 96).

See Also

MAFDS_GetValueData (page 96)

MAFDS_GetParentContainer

Returns the parent container DN of the specified object.

Syntax

C

```
#include <mafds.h>

NMASAPI int MAFDS_GetParentContainer
(
    MAFDS_CONTEXT     context,
    unicode          *objectDN,
    unicode          *parentDN,
    nuint32           parentDNSize
);
```

Parameters

`context`

(IN) A MAFDS context allocated using MAFDS_CreateContext (page 88).

`objectDN`

(IN) The object whose parent is to be retrieved. The DN should be provided in typeless, dot-delimited format (for example, `Tom.Accounting.AcmeCorp`).

`parentDN`

(OUT) A buffer to receive the DN of the parent container.

`parentDNSize`

(IN) The size of the buffer supplied in the "parentDN" parameter.

Return Codes

Table 4-3 *MAFDS_GetParentContainer Return Codes*

Value	Description
NMAS_E_ENTRY_NOT_FOUND	The object specified in "objectDN" does not exist.
NMAS_E_BUFFER_OVERFLOW	The buffer pointed to by "parentDN" is too small.

See Also

MAFDS_CreateContext (page 88)

MAFDS_GetPartitionRootContainer

Returns the partition root container DN of the specified object.

Syntax

C

```
#include <mafds.h>

NMASAPI int MAFDS_GetPartitionRootContainer
(
    MAFDS_CONTEXT    context,
    unicode         *objectDN,
    unicode         *partRootDN,
    nuint32          partRootDNSize
);
```

Parameters

`context`

(IN) A MAFDS context allocated using MAFDS_CreateContext (page 88).

`objectDN`

(IN) The object whose partition root container is to be retrieved.

`partRootDN`

(OUT) A buffer to receive the DN of the partition root container. The DN should be provided in typeless, dot-delimited format (for example, `Tom.Accounting.AcmeCorp`).

`partRootDNSize`

(IN) The size of the buffer supplied in the "parentDN" parameter.

Return Codes

Table 4-4 *MAFDS_GetParentContainer Return Codes*

Value	Description
NMAS_E_ENTRY_NOT_FOUND	The object specified in "objectDN" does not exist.
NMAS_E_BUFFER_OVERFLOW	The buffer pointed to by "partRootDN" is too small.

See Also

MAFDS_CreateContext (page 88)

MAFDS_GetValueData

Retrieves individual values from a MAFDS_VALUE handle. Values must first be read from the data store using either MAFDS_ReadAttributeValues (page 102) or MAFDS_ReadInheritedAttributeValues (page 104).

Syntax

C

```
#include <mafds.h>

NMASAPI int MAFDS_GetValueData
(
    MAFDS_CONTEXT     context,
    MAFDS_CONTEXT     values,
    MAFDS_VALUE_DATA *valueData
);
```

Parameters

`context`

(IN) The context that was used to read the values.

`values`

(IN) The MAFDS_VALUE handle which contains the values to be retrieved.

`valueData`

(OUT) A pointer to a MAFDS_VALUE_DATA structure to receive the next value.

Return Codes

NMAS_E_NO_MORE_ENTRY_ATTRIBUTES: There are no more values to be retrieved from the value handle specified in “values”.

Remarks

One value will be returned on each call to MAFDS_GetValueData (page 96). To retrieve all the values from the MAFDS_VALUE handle, a login method should call MAFDS_GetValueData in a loop until it returns NMAS_E_NO_MORE_ENTRY_ATTRIBUTES.

The caller must free the contents of the MAFDS_VALUE_DATA structure using MAFDS_FreeValueData (page 93) after each successful call.

See Also

MAFDS_FreeValueData (page 93)
MAFDS_ReadAttributeValues (page 102)
MAFDS_ReadInheritedAttributeValues (page 104)

MAFDS_InsertModValue

Inserts a data store modification request into a MAFDS_MODVALUE handle. Use MAFDS_ModifyEntry (page 100) to execute the modification request.

Syntax

C

```
#include <mafds.h>

NMASAPI int MAFDS_InsertModValue
(
    MAFDS_CONTEXT      context,
    int               *operation,
    MAFDS_ATTRIBUTE   *attr,
    void              *value,
    size_t             valueSize,
    MAFDS_MODVALUE    *modValues
);
```

Parameters

context

(IN) A MAFDS context allocated using MAFDS_CreateContext (page 88).

operation

(IN) The type of modification to be performed. This may be one of the following values:

Table 4-5 MAFDS_InsertModValue Operational Parameters

Value	Description
MAFDS_MODOP_ADD_VALUE	Adds a value to an attribute. If the attribute does not exist, it will be created. If the attribute exists and it is a single-values attribute, this call will fail with an NMAS_E_ENTRY_EXISTS error.
MAFDS_MODOP_MODIFY_VALUE	Change a value for a single-values attribute, or add a new value to a multi-values attribute.
MAFDS_MODOP_DEL_VALUE	Remove a value from an attribute.
MAFDS_MODOP_DEL_ATTR	Remove an attribute and all of its values.

attr

(IN) The attribute to be modified.

value

(IN) If a new attribute value is being added, this parameter must contain the new value. This parameter may be NULL if MAFDS_MODOP_DEL_ATTR in the “operation” parameter.

valueSize

(IN) The size in bytes of the buffer pointed to by “value”.

`modValues`

(IN/OUT) The address of a handle that contains the modification request. If this is the first modification request for this handle, the handle must be pre-initialized to NULLL.

Remarks

The first call to MAFDS_InsertModValue allocates a MAFDS_MODVALUE handle to store the modification request. Subsequent calls may be made with the same MAFDS_MODVALUE handle to store additional modification requests in the handle.

The MAFDS_MODVALUE handle must be pre-initialized to NULL prior to adding the first modification request.

Modification requests may be executed using the MAFDS_ModifyEntry (page 100) call. After calling MAFDS_ModifyEntry, the login method must call MAFDS_FreeModValues (page 91) to free the MAFDS_MODVALUE handle.

See Also

MAFDS_FreeModValues (page 91)
MAFDS_ModifyEntry (page 100)

MAFDS_ListContainerEntries

Lists all of the objects in a container. The list may be filtered by object class.

Syntax

C

```
#include <mafds.h>

NMASAPI int MAFDS_ListContainerEntries
(
    MAFDS_CONTEXT    context,
    char            *dn,
    char            *objectClass,
    char          ***entries
);
```

Parameters

`context`

(IN) A MAFDS context allocated using MAFDS_CreateContext.

`dn`

(IN) The DN of the container whose contents are to be listed. The DN should be provided in typeless, dot-delimited format (for example, `ou=Accounting,o=AcmeCorp`)

`objectClass`

(IN) The class of the objects to be listed. If this parameter is NULL then all objects will be listed.

`entries`

(OUT) The address of a NULL-terminated array of strings containing the DN of all objects that were found in the search.

Return Codes

NMAS_E_ENTRY_NOT_FOUND: The container specified in “dn” does not exist.

Remarks

On successful completion, the “entries” parameter will contain the requested object list. The caller must free this list using MAFDS_FreeContainerEntries (page 89).

MAFDS_ModifyEntry

Execute all of the modification requests stored in a MAFDS_MODVALUE handle. Modification requests may be added to a MAFDS_MODVALUE handle using MAFDS_InsertModValue (page 97).

Syntax

C

```
#include <mafds.h>

NMASAPI int MAFDS_ModifyEntry
(
    MAFDS_CONTEXT   context,
    char           *dn
    MAFDS_MODVALUE *modValues
);
```

Parameters

`context`

(IN) A MAFDS context allocated using MAFDS_CreateContext (page 88).

`dn`

(IN) The DN of the object to be modified. The DN should be provided in typeless, dot-delimited format (for example, `Tom.Accounting.AcmeCorp`).

`modValues`

(IN) A MAFDS_MODVALUE handle that contains one or more modification request to be executed.

Return Codes

Table 4-6 *MAFDS_ModifyEntry Return Codes*

Value	Description
NMAS_E_ENTRY_NOT_FOUND	The container specified in “dn” does not exist
NMAS_E_ENTRY_EXISTS	A request was made to add an additional value to an existing single-valued attribute, or to add a duplicate value to a multi-valued attribute.

Remarks

This call executes the modification requests stored in a MAFDS_MODVALUE handle. Requests may be added to a MAFDS_MODVALUE handle using the MAFDS_InsertModValue (page 97) call. The login method is responsible for freeing the MAFDS_MODVALUE handle using MAFDS_FreeModValues (page 91).

If this call fails because one of the requested modifications could not be completed, then none of the requested modifications will be performed.

See Also

MAFDS_CreateContext (page 88)
MAFDS_FreeModValues (page 91)
MAFDS_InsertModValue (page 97)

MAFDS_ReadAttributeValues

Retrieves the values of the specified attributes on an object. Call MAFDS_GetValueData (page 96) to retrieve data read by this call.

Syntax

C

```
#include <mafds.h>

NMASAPI int MAFDS_ReadAttributeValues
(
    MAFDS_CONTEXT      context,
    unicode           *dn,
    MAFDS_ATTRIBUTE   *attrs,
    nuint32            numAttrs,
    MAFDS_VALUE       *values,
    nuint32           *numValuesRead
);
```

Parameters

context

(IN) The context to use for the read operation.

dn

(IN) The DN of the object whose attributes are to be read. The DN should be provided in typeless, dot-delimited format (for example, `Tom.Accounting.AcmeCorp`).

attrs

(IN) An array of MAFDS_ATTRIBUTE structures which specify the attributes to be read.

numAttrs

(IN) The number of elements in `attrs`.

values

(OUT) A handle containing the requested values. Call MAFDS_GetValueData (page 96) to retrieve the values from this handle.

numValuesRead

(OUT) The number of values retrieved from the data store. This parameter may be NULL.

Return Codes

Table 4-7 *MAFDS_ReadAttributeValues Return Codes*

Value	Description
NMAS_E_ENTRY_ATTRIBUTE_NOT_FOUND	One or more of the specified attributes does not exist on the specified object.
NMAS_E_ENTRY_NOT_FOUND	The specified object does not exist.

See Also

MAFDS_GetValueData (page 96)

MAFDS_ReadInheritedAttributeValues

Reads the specified attribute. Call MAFDS_GetValueData (page 96) to retrieve data read by this call.

This call attempts to read an attribute first from the user object, then the parent container, then the partition root container. The call will return as soon as the attribute is found. For example, if the attribute is found on the user object, then this call will not attempt to read the parent container or the partition root.

Syntax

C

```
#include <mafds.h>

NMASAPI int MAFDS_ReadInheritedAttributeValues
(
    MAFDS_CONTEXT      context,
    unicode           *objectDN,
    MAFDS_ATTRIBUTE   *attr,
    MAFDS_VALUE       *values,
    nuint32           *numValuesRead,
    unicode           *objectRead,
    nuint32            objectReadSize
);
```

Parameters

`context`

(IN) The context to use for the read operation.

`objectDN`

(IN) The DN of the object whose attributes are to be read. The DN should be provided in typeless, dot-delimited format (for example, `Tom.Accounting.AcmeCorp`).

`attr`

(IN) The attribute to be read.

values

(OUT) A handle containing the requested values. Call MAFDS_GetValueData (page 96) to retrieve the values from this handle.

numValuesRead

(OUT) A pointer to an integer that will contain the number of values read from the data store on a successful return. This parameter may be NULL.

objectRead

(OUT) A pointer to a buffer that will receive the DN of the object from which the values were read.

objectReadSize

(IN) The size in bytes of the buffer pointed to by the objectRead parameter.

Return Codes

Table 4-8 *MAFDS_ReadInheritedAttributeValues Return Codes*

Value	Description
NMAS_E_BUFFER_OVERFLOW	The buffer supplied in the objectRead parameter was too small.
NMAS_E_ENTRY_ATTRIBUTE_NOT_FOUND	The specified attribute does not exist on the specified object, its parent container, or on the partition root container.
NMAS_E_ENTRY_NOT_FOUND	The specified object does not exist.

See Also

MAFDS_GetValueData (page 96)

MAFDS_VALUE_DATA

Contains an individual attribute value.

Structure

C

```
typedef struct MAFDS_VALUE_DATA
{

    void       *data;
    size_t      size;
    unicode    *attr;
    int         syntax;
};
```

Fields

data

The attribute value. The format of this field depends on the attribute syntax.

size

The size of the buffer pointed to by the “data” field.

attr

The attribute from which this value was read.

syntax

The syntax of the attribute. (See “MAFDS Syntax Constants” on page 53 for a listing of possible values.)

5 Method Management Functions

The NMAS Method Management functions are used to manage the configuration and secret data for a login method. These functions are typically used by the by a login method management tool. An iManager plugin is one example that reads data from a smart card and assigns that card to a user so they can authenticate using a smart card login method. This section contains the following topics:

- Section 5.1, "Configuration Store Functions," on page 107
- Section 5.2, "SecretStore Management Functions," on page 107
- Section 5.3, "Login Method Management Functions," on page 108

5.1 Configuration Store Functions

Configuration data is stored in the Authentication Store in eDirectory™. It can contain any user-defined data and can be accessed and viewed using the Method Management functions. Data in the Configuration Store, unlike Novell SecretStore™ data, can be retrieved and viewed through the MMG interface program.

Configuration data is secured in eDirectory with a NICI encryption. The configuration data storage area is defined as a multiple value attribute that can be associated with any user object, Login Method object, or Login Device object within eDirectory. Data can also be read from the Authentication Store using the MAF_GetAttribute (page 59) function or modified using MAF_PutAttribute (page 72).

The login configuration functions are defined below:

- nmasldap_delete_login_config (page 119)
- nmasldap_get_login_config (page 123)
- nmasldap_put_login_config (page 125)

5.2 SecretStore Management Functions

The MMG functions provide the ability to write but not read Secret data. Secret data can only be read by the LSM through the MAF_GetAttribute (page 59) function. Secret data is secured with an authentication strength encryption. The secret data storage area is defined as a multiple value attribute that can be associated with any user object, Login Method object, or Login Device object within eDirectory.

The SecretStore management functions are defined below:

- nmasldap_delete_login_secret (page 121)
- nmasldap_put_login_secret (page 127)

5.3 Login Method Management Functions

The NMAS APIs include a set of new LDAP functions for both C and Java. The Java library supports both JNDI and JLDAP. The functions/methods are grouped below by operation. All of the java methods are contained in the com.novell.security.nmas.mgmt.NMASLoginDataMgr class, in the NMASToolkit.jar.

For more information about Java password management functions, see Section 7.1, "NMAS Password Management Java Classes," on page 135.

We also provided the original Windows-Only Legacy Functions, which are written in C and Java. We recommend that all future development be based on the new LDAP functionality. The following lists all of the login method management functions included in this API:

- See the LDAP Java Methods (../api/index.html)
  - deleteLoginConfig (../api/com/novell/security/nmas/mgmt/NMASLoginDataMgr.html#deleteLoginConfig)
  - deleteLoginSecret (../api/com/novell/security/nmas/mgmt/NMASLoginDataMgr.html#deleteLoginSecret)
  - getLoginConfig (../api/com/novell/security/nmas/mgmt/NMASLoginDataMgr.html#getLoginConfig)
  - putLoginConfig (../api/com/novell/security/nmas/mgmt/NMASLoginDataMgr.html#putLoginConfig)
  - putLoginSecret (../api/com/novell/security/nmas/mgmt/NMASLoginDataMgr.html#putLoginSecret)
- Windows-Only Legacy Functions
  - NMAS_DeleteLoginConfig (page 109)
  - NMAS_DeleteLoginSecret (page 110)
  - NMAS_GetLoginConfig (page 111)
  - NMAS_PutLoginConfig (page 115)
  - NMAS_PutLoginSecret (page 117)

IMPORTANT: The NMAS Login Data Management API requires NMAS 2.1 or later. LDAP extensions are used to communicate with eDirectory. An authenticated SSL connection is required for all APIs.

NMAS_DeleteLoginConfig

Deletes the data associated with the specified tag and method on the named object. This function is available to the Novell Client and server. These interfaces are only available to Windows applications.

Syntax

C

```
#include <nmas.h>
#include <nmasext.h>


nint   NMAS_DeleteLoginConfig
(
   unicode    *treeName
   unicode    *objectDN,
   nuint32     methodIDLen,
   pnuint32    methodID,
   unicode    *tag
);
```

Parameters

`treeName`

(IN) The NDS® eDirectory tree name.

`objectDN`

(IN) Identifies the object that holds the login data. dot-delimited typeless DN.
C. Unicode string

`methodIDLen`

(IN) The number of bytes in the methodID. The only valid value is 4.

`methodID`

(IN) Unique method identifier. A 32-bit unsigned integer.

`tag`

(IN) Tag associated with the data. C: Unicode string

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_GetLoginConfig (page 111)

NMAS_DeleteLoginSecret

Deletes the login secret data associated with the specified tag and method on the named object. This function is available to the Novell Client and server. These interfaces are only available on Windows platforms.

Syntax

C

```
#include <nmas.h>
#include <nmasext.h>


nint   NMAS_DeleteLoginSecret
(
   unicode    *treeName
   unicode    *objectDN,
   nuint32     methodIdLen,
   pnuint32    methodID,
   unicode    *tag
);
```

Parameters

treeName

(IN) The eDirectory tree name.

objectDN

(IN) Identifies the object that holds the login data. dot-delimited typeless DN.
C: Unicode string

methodIDLen

(IN) The number of bytes in the methodID. Valid values are 4 or 16.

methodID

(IN) Unique method identifier. A 32-bit unsigned integer.

tag

(IN) Tag associated with the data.

C: Unicode string

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_PutLoginSecret (page 117)

NMAS_GetLoginConfig

Returns the data specified by the method tag from the Login Configuration store for the specified object. The data is decrypted before it is returned. The maximum size of the login configuration and login secret data is 60,000 bytes. This function is available to the Novell Client and server. These interfaces are only available on Windows platforms.

Syntax

C

```
#include <nmas.h>
#include <nmasext.h>

nint   NMAS_GetLoginConfig
(
   unicode   *treeName
   unicode   *objectDN,
   nuint32    methodIDLen,
   pnuint32   methodID,
   unicode   *tag,
   size_t     bufferLen,
   size_t    *dataLen,
   void      *data
);
```

Parameters

treeName

(IN) Identifies the eDirectory tree name.

objectDN

(IN) Identifies the object that holds the login data. dot-delimited typeless DN.

C: Unicode string

methodIDLen

(IN) The number of bytes in the methodID. The only valid value is 4.

methodID

(IN) Unique method identifier. A 32-bit unsigned integer.

tag

(IN) Tag associated with the data.

C: Unicode string

bufferLen

(IN) Size in bytes of the buffer.

dataLen

(OUT) Size in bytes of the data returned.C: pnint32

data

(OUT) Requested data.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_PutLoginConfig (page 115)
MAF_GetAttribute (page 59)
MAF_PutAttribute (page 72)

NMAS_Login

Performs an NMAS login sequence.

Syntax

C

```
#include <nmas.h>
#include <nmasclnt.h>
#include <nmasext.h>

N_EXTERN_LIBRARY (int) NMAS_Login
(
   nstr8     *pTreeName,
   nstr8     *pUserDN,
   nstr8     *pPwd,
   nstr8     *pServer,
   nstr8     *pSequence,
   nstr8     *pClearance,
   nptr       uiHandle,
   nuint32    uiTimeout,
   nuint32    options
);
```

Parameters

`TreeName`

(IN) The eDirectory tree in which to login.

`pUserDN`

(IN) Specifies the dot delimited distinguished name of the user to login.

`pPwd`

(IN) The password for login method, which can be NULL for methods that don't use a password.

`pServer`

(IN) The name of the server in the eDirectory tree to use for login. If specified, the API will attempt to use the specific server for login. However, if the specified server can't complete the login, a different server may be selected. This can be used to target a specific server for login.

`pSequence`

(IN) Specifies the NMAS login sequence to run. If NULL is specified, the user's default sequence is used.

`pClearance`

(IN) Specifies the NMAS login clearance. Specify NULL if the clearance is not used.

`uiHandle`

(Optional) (IN) On Windows platforms, this is the parent window handle that can be use by methods that display a user interface. This allows the method to create windows with a proper parent/child window relationship. Specify NULL if your method does not use this parameter.

uiTimeout

(IN) Specifies the user interface inactivity timeout (in seconds) passed to methods. A value of 0 indicates no timeout. If a method displays a user interface, it can use this value to timeout its windows due to inactivity.

options

(IN) Specifies one of NMAS_OPT_*xxx* values defined in `nmasclnt.h`.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_GetLoginConfig (page 111)
MAF_GetAttribute (page 59)
MAF_PutAttribute (page 72)

NMAS_PutLoginConfig

Stores the tag and the data in the Login Configuration store for the specified method and specified object. The data is encrypted using the data strength encryption key associated with the Login Configuration store. The maximum size of the login configuration and login secret data is 60,000 bytes. This function is available to the Novell Client and server. These interfaces are only available on Windows platforms.

Syntax

C

```
#include <nmas.h>
#include <nmasext.h>

nint   NMAS_PutLoginConfig
(
   unicode   *treeName
   unicode   *objectDN,
   nuint32    methodIDLen,
   pnuint32   methodID,
   unicode   *tag,
   size_t     dataLen,
   void      *data
);
```

Parameters

treeName

(IN) The eDirectory tree name.

objectDN

(IN) Identifies the object that holds the login data. dot-delimited typeless distinguished name.

C: Unicode string

methodIDLen

(IN) The number of bytes in the methodID. The only valid value is 4.

methodID

(IN) Unique method identifier. A 32-bit unsigned integer.

NOTE: This is the method identification number of the method that is allowed to access the data. A method identification number of zero means that the data can be accessed by any method by using the NMAS_AID_USER_GLOBAL_CONFIG_DATA AID (see Section 4.2, "NMAS Attribute IDs," on page 47).

tag

(IN) Tag associated with the data. C: Unicode string

dataLen

(IN) Size in bytes of the data.

`data`

(IN) Data to be encrypted and stored in the object.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_GetLoginConfig (page 111)
MAF_GetAttribute (page 59)
MAF_PutAttribute (page 72)

NMAS_PutLoginSecret

Stores the tag and the data in the Login Secret store for the specified method and object. The data is encrypted using the authentication strength encryption key associated with the Login Secret store. This function is available to the Novell Client and server. These interfaces are only available on Windows platforms.

Syntax

C

```
#include <nmas.h>
#include <nmasext.h>

 nint   NMAS_PutLoginSecret
(
    unicode   *treeName
    unicode   *objectDN,
    nuint32    methodIdLen,
    pnuint32   methodID,
    unicode   *tag,
    size_t     dataLen,
    void      *data
);
```

Parameters

treeName

(IN) The eDirectory tree name.

objectDN

(IN) Identifies the object that holds the login data. dot-delimited typeless DN.

C: Unicode string

methodIDLen

(IN) The number of bytes in the methodID. The only valid value is 4.

methodID

(IN) Unique method identifier. A 32-bit unsigned integer.

tag

(IN) Tag associated with the data.

C: Unicode string

dataLen

(IN) Size in bytes of the data.

data

(IN) Data to be encrypted and stored in the object.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

Remarks

Server APIs. There is no *NMAS_GetLoginSecret* function because the secret data can only be read by an LSM.

See Also

MAF_GetAttribute (page 59)

nmasldap_delete_login_config

Deletes the data associated with the specified tag and method on the named object. This function is typically used by management applications and is provided in the NMAS “ldapext” shared library.

Syntax

C

```
#include <nmas.h>
#include <nmasext.h>

 int nmasldap_delete_login_config
(
    LDAP        *ld,
    char      *objectDN,
    nuint32    methodIDLen,
    pnuint32   methodID,
    char      *tag
);
```

Parameters

ld

(IN) An authenticated LDAP connection.

objectDN

(IN) Identifies the object that holds the login data. dot-delimited typeless DN.

C: Unicode string

methodIDLen

(IN) The number of bytes in the methodID. The only valid value is 4.

methodID

(IN) The unique method identifier assigned to a login method. The method identifier specifies to which NMAS login method the login secret or configuration data is associated. The zero method identifier zero (0) indicates that the login secret data can be accessed by any login method. Examples of assigned method identifiers include the following:

- **0x00000000:** Global login method data available to all login methods
- **0x00000007:** NDS Password Login Method.
- **0x00000009:** Simple Password Login Method
- **0x0000000B:** EXTERNAL SASL Mechanism
- **0x0000001C:** DIGEST-MD5 SASL Mechanism
- **0x0000001F:** Challenge/Response Login Method
- **0x00000025:** Novell Enhanced Smartcard Login Method

tag

(IN) Tag associated with the data.

C: Unicode string

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

- An NMAS LDAP Extension version number
- A BER Value that contains the input unicode target object eDirectory DN
- A BER Value that contains the input method identifier
- A BER Value that contains the input unicode tag
- The input data as a bitstring

See Also

nmasldap_get_login_config (page 123)
nmasldap_put_login_config (page 125)

nmasldap_delete_login_secret

Deletes the login secret data associated with the specified tag and method on the named object and is typically used by management.

Syntax

C

```
#include <nmas.h>
#include <nmasext.h>

 int nmasldap_delete_login_secret
(
    LDAP      *ld,
    Unicode   *objectDN,
    size_t     methodIDLen,
    pnuint32   methodID,
    unicode   *tag
);
```

Parameters

`ld`

(IN) An authenticated LDAP conection.

`objectDN`

(IN) Identifies the object that holds the login data. dot-delimited typeless DN.
C: Unicode string

`methodIDLen`

(IN) The number of bytes in the methodID. Valid values are 4 or 16.

`methodID`

(IN) The unique method identifier assigned to a login method. The method identifier specifies to which NMAS login method the login secret data is associated. The zero method identifier zero (0) indicates that the login secret data can be accessed by any login method. Examples of assigned method identifiers include the following:

- **0x00000000:** Global login method data available to all login methods
- **0x00000007:** NDS Password Login Method
- **0x00000009:** Simple Password Login Method
- **0x0000000B:** EXTERNAL SASL Mechanism
- **0x0000001C:** DIGEST-MD5 SASL Mechanism
- **0x0000001F:** Challenge/Response Login Method
- **0x00000025:** Novell Enhanced Smartcard Login Method

`tag`

(IN) Tag associated with the data.

C: Unicode string

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

nmasldap_delete_login_secret (page 121)

nmasldap_get_login_config

Returns the data specified by the method tag from the Login Configuration store for the specified object. The data is decrypted before it is returned. The maximum size of the login configuration and login secret data is 60,000 bytes. This function is typically used by management applications and is provided in the NMAS “ldapext” shared library.

Syntax

C

```
#include <nmas.h>
#include <nmasext.h>

int nmasldap_get_login_config
(
   LDAP        *ld,
   char       *objectDN,
   nuint32     methodIDLen,
   pnuint32    methodID,
   char       *tag,
   size_t     *dataLen,
   void       *data
);
```

Parameters

ld

(IN) An authenticated LDAP conection.

objectDN

(IN) Identifies the object that holds the login data. dot-delimited typeless DN.

C: Unicode string

methodIDLen

(IN) The number of bytes in the methodID. The only valid value is 4.

methodID

(IN) The unique method identifier assigned to a login method. The method identifier specifies to which NMAS login method the login secret or configuration data is associated. The zero method identifier zero (0) indicates that the login secret data can be accessed by any login method. Examples of assigned method identifiers include the following:

- **0x00000000:** Global login method data available to all login methods
- **0x00000007:** NDS Password Login Method.
- **0x00000009:** Simple Password Login Method
- **0x0000000B:** EXTERNAL SASL Mechanism
- **0x0000001C:** DIGEST-MD5 SASL Mechanism
- **0x0000001F:** Challenge/Response Login Method
- **0x00000025:** Novell Enhanced Smartcard Login Method

`tag`

(IN) Tag associated with the data.

C: Unicode string

`dataLen`

(OUT) Size in bytes of the data returned.C: pnint32

`data`

(OUT) Requested data.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

nmasldap_delete_login_config (page 119)
nmasldap_put_login_config (page 125)

nmasldap_put_login_config

Stores the tag and the data in the Login Configuration store for the specified method and specified object. The data is encrypted using the data strength encryption key associated with the Login Configuration store. The maximum size of the login configuration and login secret data is 60,000 bytes. This function is typically used by management applications and is provided in the NMAS “ldapext” shared library.

Syntax

C

```
#include <nmas.h>
#include <nmasext.h>

 int nmasldap_put_login_config
(
    LDAP       *ld,
    char      *objectDN,
    nuint32    methodIDLen,
    pnuint32   methodID,
    char      *tag,
    size_t     dataLen,
    void      *data
);
```

Parameters

ld

(IN) An authenticated LDAP conection.

objectDN

(IN) Identifies the object that holds the login data. dot-delimited typeless DN.

C: Unicode string

methodIDLen

(IN) The number of bytes in the methodID. The only valid value is 4.

methodID

(IN) The unique method identifier assigned to a login method. The method identifier specifies to which NMAS login method the login secret or configuration data is associated. The zero method identifier zero (0) indicates that the login secret data can be accessed by any login method. Examples of assigned method identifiers include the following:

- **0x00000000:** Global login method data available to all login methods
- **0x00000007:** NDS Password Login Method.
- **0x00000009:** Simple Password Login Method
- **0x0000000B:** EXTERNAL SASL Mechanism
- **0x0000001C:** DIGEST-MD5 SASL Mechanism
- **0x0000001F:** Challenge/Response Login Method
- **0x00000025:** Novell Enhanced Smartcard Login Method

`tag`

(IN) Tag associated with the data.

C: Unicode string

`dataLen`

(IN) Size in bytes of the data.

`data`

(IN) Data to be encrypted and stored in the object.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

nmasldap_delete_login_config (page 119)
nmasldap_get_login_config (page 123)

nmasldap_put_login_secret

Stores the tag and the data in the Login Secret store for the specified method and object. The data is encrypted using the authentication strength encryption key associated with the Login Secret store. This function is typically used by management applications and is provided in the NMAS “ldapext” shared library.

Syntax

C

```
#include <nmas.h>
#include <nmasext.h>

 int nmasldap_put_login_secret
(
    LDAP        *ld,
    char      *objectDN,
    nuint32    methodIDLen,
    pnuint32   methodID,
    char      *tag,
    size_t     dataLen,
    void      *data
);
```

Parameters

`ld`

(IN) An authenticated LDAP conection.

`objectDN`

(IN) Identifies the object that holds the login data. dot-delimited typeless DN.

C: Unicode string

`methodIDLen`

(IN) The number of bytes in the methodID. The only valid value is 4.

`methodID`

(IN) The unique method identifier assigned to a login method. The method identifier specifies to which NMAS login method the login secret data is associated. The zero method identifier zero (0) indicates that the login secret data can be accessed by any login method. Examples of assigned method identifiers include the following:

- **0x00000000:** Global login method data available to all login methods
- **0x00000007:** NDS Password Login Method
- **0x00000009:** Simple Password Login Method
- **0x0000000B:** EXTERNAL SASL Mechanism
- **0x0000001C:** DIGEST-MD5 SASL Mechanism
- **0x0000001F:** Challenge/Response Login Method
- **0x00000025:** Novell Enhanced Smartcard Login Method

`tag`

(IN) Tag associated with the data.

C: Unicode string

`dataLen`

(IN) Size in bytes of the data.

`data`

(IN) Data to be encrypted and stored in the object.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

nmasldap_delete_login_secret (page 121)

6 NMAS Login Policy Management

This section describes the NMAS login policy management functions and the extensions in the following topics:

6.1 NMAS Login Policy Management

NMAS Login Policy Management C Functions

nmasldap_policy_refresh

Requests that NMAS re-reads the login policy, and re-loads all login methods.

Syntax

C

```
#include <nmas.h>
#include <nmasext.h>

int int nmasldap_policy_refresh
(
    LDAP          #ld,
);
```

Parameters

ld

(IN) The LDAP session handle.

Return Codes

Returns 0 if successful; not equal to 0 if not successful and an NMAS error code is returned.

See Also

nmasldap_check_login_policy

Queries NMAS to determine if the user's login policy will allow the user to log in. Also, depending upon the specified flags, this request will update the user login attributes, such as login intruder attributes, login failure time, login time, last login time, and remaining grace logins.

Syntax

C

```
#include <nmas.h>
#include <nmasext.h>

int nmasldap_check_login_policy
(
    LDAP            *ld,
    char            *objectDN,
    unsigned int    flags,
    size_t          netAddressSize,
    unsigned char *netAddress
);
```

Parameters

`ld`

(IN) The LDAP session handle.

`objectDN`

(IN) Identifies the target object. An LDAP DN.

C: utf-8 string

`flags`

(IN) A combination of the following flags can be used to specify the login policy checks performed and login attributes that will be updated.

LOGIN_POLICY_CHECK (0x1) Only checks if the specified user is allowed to log in.

LOGIN_POLICY_SUCCESS_UPDATE (0x2) Updates the user's login attributes as if login were successful

LOGIN_POLICY_FAILURE_UPDATE (0X4) Updates the user's login attributes as if login failed.

LOGIN_POLICY_PWD_POLICY_CHECK (0x8) Checks if the user is allowed to login using the password.

`netAddressSize`

(IN) Specifies the size of the network address provided in the netAddress parameter.

`netAddress`

(IN) Designates the network address that is used to determine if the user is allowed to login. The first character specifies the type of network address where '0' specifies an IPX address and '1' specifies an IP address.

The following bytes specify the network address.

For example, if the network address is the IP address 12.34.56.78 then netAddress would be set to {'1', '#', 12, 34, 56, 78, 0}

If the network address is the IPX address 112233441122334455661122 then netAddress would be set to [‘0’, ‘#’, 0x11, 0x22, 0x33, 0x44, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x11, 0x22, 0}

C: UTF-8 string

Return Codes

Returns 0 if successful; not equal to 0 if not successful and an NMAS error code is returned.

See Also

nmasldap_set_address_policy (page 133)

nmasldap_set_address_policy

Adds a login network address restriction to the target object. Network restrictions indicate which network address a user can log in from.

Syntax

C

```
#include <nmas.h>
#include <nmasext.h>

int nmasldap_set_address_policy
(
    LDAP      *ld,
    char      *objectDN,
    nuint32    flags,
    size_t     netAddressSize,
    nuint8    *netAddress
);
```

Parameters

ld

(IN) The LDAP session handle.

objectDN

(IN) Identifies the target object. An LDAP DN.

C: UTF-8 string

flags

(IN) Indicates if the specified network address is to be added to the list of restricted network addresses, or if the specified network address is to be removed from the list of restricted addresses.

LOGIN_POLICY_ADD_RESTRICTION (0x1) The specified network address will be added to the list of restricted network addresses.

LOGIN_POLICY_RM_RESTRICTION (0x2) The specified network address will be removed from the list of restricted network addresses.

netAddressSize

(IN) Specifies the size of the network address provided in the netAddress parameter.

netAddress

(IN) Designates network addresses from which the user can log in. The first character specifies the type of network address where ‘0’ specifies an IPX address and ‘1’ specifies an IP address.

The following bytes specify the network address:

For example, if the network address is the IP address 12.34.56.78 then netAddress would be set to {‘1’, ‘#’, 12, 34, 56, 78, 0}

If the network address is the IPX address 1122334411223344556611 22 then netAddress would be set to [‘0’, ‘#’, 0x11, 0x22, 0x33, 0x44, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x11, 0x22, 0}

C: UTF-8 string

Return Codes

Returns 0 if successful; not equal to 0 if not successful and an NMAS error code is returned.

See Also

nmasldap_check_login_policy (page 131)

7 Password Management Functions

This section describes the NMAS password management functions and extensions in the following topics:

7.1 NMAS Password Management Java Classes

The new Java classes now available in NMAS NDK are consumed by Novell iManager and other applications that need to control the data required by simple password and universal password login methods.

While the previous NMAS API functions made native calls on the workstation, the new Java classes require no native components. We strongly recommend that you use the new LDAP extensions for all new NMAS development. If you want to administer your login methods through iManager or some other similar management utility, consider using the Java classes. All of the required Java classes are located in the `JavaToolkit.jar` file.

The Java library includes two interfaces, which can be selected depending on the type of LDAP connection you have to your LDAP server:

- **JNDI:** Uses an LDAP context object and requires no additional libraries. The JNDI interface was implemented first and can be used by applications with no additional `.jar` files besides `NMASToolkit.jar`.
- **JLDAP:** A proprietary Novell library that provides an LDAP connection. If you use this interface, you will also require an `ldap.jar` file, which is part of the NMAS NDK download package (http://www.novell.com/developer/ndk/novell_modular_authentication_service.html).

The two constructors look like this:

```
public NMASPwdMgr(LdapContext ldapCtx)
{    this.transport = new PwdLdapTransport(ldapCtx);     }     /**
     * @param pwdTransport  A PwdTransport object initialized
     * with the appropriate connection.
     */     public NMASPwdMgr(PwdTransport pwdTransport)
{    this.transport = pwdTransport;     }
```

With the second constructor, the calling application must instantiate the transport object and pass it in.

Whenever you implement JNDI applications, an extra ldap.jar file is not required to build or deploy it because it remains an unused constructor in a class in a `.jar` file.

To review all of the Java classes delivered in the NMAS NDK, see the Chapter 9, "NMAS Javadoc References," on page 171.

7.2 Password Management Requirements

Before implementing the NMAS management functions, make sure your development environment contains the following files:

- **C Development:** `nmasext.h`, `nmasPkt.h`, `ldap.h`.
- **Java Development:** com.novell.security.nmas.mgmt.`NMASLoginDataMgr.java`, com.novell.`security.nmas.mgmt.NMASPwdMgr.java`, and `com.novell.security.nmas.mgmt.NMASSimplePwdMgr.java`. These classes provide the highest level interface to the NMAS management API. These classes also access the lower level "request" and "response" classes. The encoding and decoding of data is handled in the "BerEncoding/Decoding" classes.

7.2.1 Other Development Requirements

Before implementing the NMAS LDAP password management routines, you should have a thorough understanding of the following concepts:

- **Secure Socket Layer (SSL):** The NMAS LDAP APIs are dependent upon over-the-wire, SSL technology. For more information about SSL, see the SSL security topics in NDK: LDAP Libraries for C.
- **Java SSL:** When you connect with a server using an SSL connection, LDAP will accept data even if it's not formatted for SSL. However, in order for an NMAS module to recognize and handle data requests, transmitted data must be encrypted using SSL.

7.3 Simple Password Management

Simple Password Management C Functions:

- nmasldap_put_simple_pwd (page 139)
- nmasldap_delete_simple_pwd (page 140)
- nmasldap_get_simple_pwd (page 141)

The simple password management functions include the following (for more complete descriptions, see NMAS Javadoc References (../api/index.html)):

- deleteSimplePwd
- getSimplePwd
- isSimplePwdAssigned
- getSimplePwd

A viewable simple password management example is available at the Novell Modular Authentication Services Sample Code (../../../samplecode/nmas_sample/index.htm) page.

7.4 Universal Password Management

The universal password management functions and methods include the following:

- "Password Management C Functions:" on page 137
- "Universal Password Management Java Methods" on page 137
- "Universal Password Management Example" on page 137

These functions call the standard ldap_extended_operation with the OID assigned for each request.

The NMAS LDAP extension handler receives the request, then calls the appropriate NMAS Server function to perform the requested operation. The handler then receives a reply from the NMAS Server function and will send the reply back to the client.

Password Management C Functions:

- nmasldap_change_password (page 142)
- nmasldap_set_password (page 143)
- nmasldap_delete_password (page 144)
- nmasldap_get_password (page 145)
- nmasldap_get_password_policy_dn (page 146)
- nmasldap_policy_check_current_password (page 147)
- nmasldap_policy_check_password (page 148)
- nmasldap_get_password_status (page 149)
- nmasldap_get_password_status_ex (page 151)
- nmasldap_get_user_random_password (page 153)
- nmasldap_get_random_password (page 154)

Universal Password Management Java Methods

Access the following Universal Password Management methods defined in the NMASPwdMgr (../api/com/novell/security/nmas/mgmt/NMASPwdMgr.html) class (see all NMAS Javadoc References (../api/index.html)):

- changePwd
- deletePwd
- getPwd
- getPwdPolicyDN
- getPwdStatus
- pwdPolicyCheck
- setPwd

Universal Password Management Example

A viewable universal password management example is available at the Novell Modular Authentication Services Sample Code (../../../samplecode/nmas_sample/index.htm) page.

7.5 NMAS LDAP C Password Management Functions

The NMAS LDAP C functions are described in this section. Viewable password management examples are available at the Novell Modular Authentication Services Sample Code (../../../samplecode/nmas_sample/index.htm) page.

nmasldap_put_simple_pwd

Sets the simple password for the object provided by the passed-in Distinguished Name.

Syntax

C

```
#include <ldap.h>
#include <ntypes.h>
#include <nmasext.h>

int   nmasldap_put_simple_pwd
(
   LDAP     *ld,
   char     *objectDN,
   char      pwd
);
```

Parameters

ld

(IN) The LDAP session handle.

objectDN

(IN) Identifies the object that holds the login data. An LDAP DN.
C: UTF-8 string

pwd

(IN) The password string.

Return Codes

Returns 0 if successful; not equal to 0 if not successful and an NMAS error code is returned.

See Also

nmasldap_delete_simple_pwd (page 140), and nmasldap_get_simple_pwd (page 141)

nmasldap_delete_simple_pwd

Deletes a simple password from the object provided by the passed-in Distinguished Name.

Syntax

C

```
#include <ldap.h>
#include <ntypes.h>
#include <nmasext.h>

int   nmasldap_delete_simple_pwd
(
   LDAP     *ld,
   char     *objectDN
);
```

Parameters

ld

(IN) The LDAP session handle.

objectDN

(IN) Identifies the object that holds the login data. An LDAP DN.
C: UTF-8 string

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

nmasldap_put_simple_pwd (page 139) and nmasldap_get_simple_pwd (page 141)

nmasldap_get_simple_pwd

Gets the simple password for the object provided by the passed-in Distinguished Name.

Syntax

C

```
#include <ldap.h>
#include <ntypes.h>
#include <nmasext.h>

int   nmasldap_get_simple_pwd
(
   LDAP        *ld,
   char       *objectDN,
   size_t      pwdLen,
   char       *pwd
);
```

Parameters

ld

(IN) The LDAP session handle.

objectDN

(IN) Identifies the object that holds the login data. An LDAP DN.
C: UTF-8 string

pwdLen

(OUT) Password length in bytes.
C: size_t int

pwd

(OUT) The password string.

C: UTF-8 string

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

nmasldap_put_simple_pwd (page 139), and nmasldap_delete_simple_pwd (page 140)

nmasldap_change_password

Changes the password of the specified object.

Syntax

C

```
#include <ldap.h>
#include <ntypes.h>
#include <nmasext.h>

int nmasldap_change_password
(
     LDAP     *ld,
     char     *objectDN,
     char     *oldPwd,
     char     *pwd
);
```

Parameters

ld

(IN) The LDAP session handle.

objectDN

(IN) Identifies the object that holds the login data. An LDAP DN.
C: UTF-8 string

oldPwd

(IN) Specifies the user's old password.
C: UTF-8 string

pwd

(IN) Specifies the user's new password.
C: UTF-8 string

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

nmasldap_set_password (page 143), nmasldap_get_password (page 145), and
nmasldap_delete_password (page 144)

nmasldap_set_password

Writes the user's new password.

Syntax

C

```
#include <ldap.h>
#include <ntypes.h>
#include <nmasext.h>

int nmasldap_set_password
(
     LDAP     *ld,
     char     *objectDN,
     char     *pwd
);
```

Parameters

ld

(IN) The LDAP session handle.

objectDN

(IN) Identifies the object that holds the login data. An LDAP DN.
C: UTF-8 string

pwd

(IN) The password that is to be set as the new password.

C: UTF-8 string

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

nmasldap_change_password (page 142), nmasldap_get_password (page 145), and nmasldap_delete_password (page 144)

nmasldap_delete_password

Deletes the password of the object specified.

Syntax

C

```
#include <ldap.h>
#include <ntypes.h>
#include <nmasext.h>

void deletePwd
(
     LDAP     *ld,
     char     *objectDN
);
```

Parameters

ld

(IN) The LDAP session handle.

objectDN

(IN) Identifies the object that holds the login data. An LDAP DN.
C: UTF-8 string

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

nmasldap_change_password (page 142), nmasldap_set_password (page 143), and nmasldap_get_password (page 145)

nmasldap_get_password

Reads the password.

Syntax

C

```
#include <ldap.h>
#include <ntypes.h>
#include <nmasext.h>

int nmasldap_get_password
(
     LDAP     *ld,
     char     *objectDN,
     size_t   *pwdLen,
     char     *pwd
);
```

Parameters

ld

(IN) The LDAP session handle.

objectDN

(IN) Identifies the object that holds the login data. An LDAP DN.
C: UTF-8 string

pwLen

(OUT) Password length in bytes.
C: size_t int

pwd

(OUT) Specifies the password.
C: UTF-8 string

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

nmasldap_change_password (page 142), nmasldap_set_password (page 143), and nmasldap_delete_password (page 144)

nmasldap_get_password_policy_dn

Follows an established password algorithm up the tree to find the password policy that relates to the specified object, then returns that password policy's distinguished name.

Syntax

C

```
#include <ldap.h>
#include <ntypes.h>
#include <nmasext.h>


int nmasldap_get_password_policy_dn
(
     LDAP     *ld,
     char     *objectDN,
     nuint    *dnSize,
     char     *dn
);
```

Parameters

`ld`

(IN) The LDAP session handle.

`objectDN`

(IN) Identifies the object that holds the login data. An LDAP DN.
C: UTF-8 string

`dnSize`

(OUT) Size of distinguished name in bytes.
C: nuint string

`dn`

(OUT) The distinguished name.
C: char string

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

nmasldap_policy_check_current_password (page 147), and nmasldap_policy_check_password (page 148)

nmasldap_policy_check_current_password

Checks a user's current password to determine if it matches the password policy that is effective for the user.

Syntax

C

```
#include <ldap.h>
#include <ntypes.h>
#include <nmasext.h>

int nmasldap_policy_check_current_password
(
     LDAP     *ld,
     char     *objectDN
);
```

Parameters

ld

(IN) The LDAP session handle.

objectDN

(IN) Identifies the object that holds the login' data. An LDAP DN.
C: UTF-8 string

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

nmasldap_get_password_policy_dn (page 146), and nmasldap_policy_check_password (page 148)

nmasldap_policy_check_password

Checks the specified password to determine if it matches the password policy that is effective for the specified user.

Syntax

C

```
#include <ldap.h>
#include <ntypes.h>
#include <nmasext.h>

int nmasldap_policy_check_password
(
     LDAP     *ld,
     char     *objectDN,
     char     *pwd
);
```

Parameters

ld

(IN) The LDAP session handle.

objectDN

(IN) Identifies the object that holds the login data. An LDAP DN.
C: UTF-8 string

pwd

(IN) (Optional) A password that is to be checked against the password policy. If this parameter is NULL the current password is checked against the password policy.

C: UTF-8 string

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

nmasldap_get_password_policy_dn (page 146), and nmasldap_policy_check_current_password (page 147)

nmasldap_get_password_status

Returns the status of the target object's Universal Password and Simple Password.

Syntax

C

```
#include <nmas.h>
#include <nmasext.h>

int nmasldap_get_password_status
(
   LDAP           *ld
   char           *objectDN
   unsigned int   *pwdStatus,
   unsigned int   *simplePwdStatus
);
```

Parameters

`ld`

(IN) The LDAP session handle

`objectDN`

(IN) Identifies the target object. An LDAP DN.

C: UTF-8 string

`pwdStatus`

(OUT) Flags that indicate the status of the target object's Universal Password. A combination of the following flag may be set.

SPM_UPWD_ENABLED (0x1) Universal Password is enabled.

SPM_UPWD_SET (0x2) Universal Password has a value.

SPM_UPWD_HISTORY_FULL (0x4) The Universal Password history is full.

SPM_UPWD_MATCHES_NDS (0x10) The Universal Password value matches the NDS Password Hash.

SPM_UPWD_OLDER_THAN_NDS (0x20) The NDS Password Hash has been set since the last time the Universal Password was set.

SPM_UPWD_MATCHES_SPWD (0x40) The Universal Password value matches the Simple Password.

`simplePwdStatus`

(OUT) Flags that indicate the status of the target object's Simple Password. A combination of the following flag may be set.

SPM_SPWD_SET (0x1) The Simple Password has a value.

SPM_SPWD_IS_CLEARTEXT (0x2) The Simple Password is not stored as a one-way hash such as SHA-1, SSHA-1, MD5, or UnixCrypt.

SPM_SPWD_MATCHES_NDS (0x10) The Simple Password value matches the NDS Password Hash.

Return Codes

Returns 0 if successful; not equal to 0 if not successful and an NMAS error code is returned.

See Also

nmasldap_get_password_status_ex (page 151)

nmasldap_get_password_status_ex

Returns the status of the target object's Universal Password and Simple Password.

Syntax

C

```
#include <ldap.h>
#include <ntypes.h>
#include <nmasext.h>

int nmasldap_get_password_status_ex
(
    LDAP          *ld,
    char          *objectDN,
    unsigned int  *serverVersion,
    unsigned int  *pwdStatus,
    unsigned int  *simplePwdStatus
);
```

Parameters

`ld`

(IN) The LDAP session handle.

`objectDN`

(IN) Identifies the target object. An LDAP DN.

C: UTF-8 string

`serverVersion`

(OUT) Indicates the version of the status data returned by the server. Possible values are the following:

NMAS_LDAP_PWD_STATUS_VERSION1 (1) Indicates that the server does not support the Distribution Password status flags or the Administrator Set Universal Password flag.

NMAS_LDAP_PWD_STATUS_VERSION2 (2) Indicates that the server supports the Distribution Password status flags and the Administrator Set Universal Password flag.

`pwdStatus`

(OUT) Flags that indicate the status of the target object's Universal Password. A combination of the following flag may be set:

SPM_UPWD_ENABLED (0x1) Universal Password is enabled.

SPM_UPWD_SET (0x2) Universal Password has a value.

SPM_UPWD_HISTORY_FULL (0x4) The Universal Password history is full.

SPM_UPWD_MATCHES_NDS (0x10) The Universal Password value matches the NDS Password Hash.

SPM_UPWD_OLDER_THAN_NDS (0x20) The NDS Password Hash has been set since the last time the Universal Password was set.

SPM_UPWD_MATCHES_SPWD (0x40) The Universal Password value matches the Simple Password.

SPM_DPWD_SET (0x100) The Distribution Password has a value.

SPM_UPWD_MATCHES_DPWD (0x200) The Universal Password value matches the Distribution Password.

SPM_UPWD_SET_BY_ADMIN (0x1000) The Universal Password was last set by someone other than the user.

`simplePwdStatus`

(OUT) Flags that indicate the status of the target object's Simple Password. A combination of the following flag may be set:

SPM_SPWD_SET (0x1) The Simple Password has a value.

SPM_SPWD_IS_CLEARTEXT (0x2) The Simple Password is not stored as a one-way hash such as SHA-1, SSHA-1, MD5, or UnixCrypt.

SPM_SPWD_MATCHES_NDS (0x10) The Simple Password value matches the NDS Password Hash.

Return Codes

Returns 0 if successful; not equal to 0 if not successful and an NMAS error code is returned.

See Also

nmasldap_get_password_status (page 149)

nmasldap_get_user_random_password

Generates and returns a random password that fulfills the requirements specified by the password policy that is effective for the target user.

Syntax

C

```
#include <nmas.h>
#include <nmasext.h>

int nmasldap_get_user_random_password
(
    LDAP      *ld,
    char      *objectDN,
    size_t    *pwdLen,
    char      *pwd
);
```

Parameters

ld

(IN) The LDAP session handle.

objectDN

(IN) Identifies the target object. An LDAP DN.

C: UTF-8 string

pwLen

(IN/OUT) Length of the random password in bytes. It should be set by the caller to the size (in bytes) of the password buffer. After this function returns, it is set to the actual size (in bytes) of the password returned in the password buffer.

C: size_t int

pwd

(OUT) Random password

C: UTF-8 string

Return Codes

Returns 0 if successful; not equal to 0 if not successful and an NMAS error code is returned.

See Also

nmasldap_get_random_password (page 154)

nmasldap_get_random_password

Generates and returns a random password that fulfills the requirements specified by the input password policy.

Syntax

C

```
#include <nmas.h>
#include <nmasext.h>

 int nmasldap_get_random_password
(
    LDAP      *ld,
    char      *xmlPwdPolicy,
    size_t    *pwdLen,
    char      *pwd
);
```

Parameters

ld

(IN) The LDAP session handle.

xmlPwdPolicy

(IN) Specifies the XML password policy that is used to generate the random password. The format of the XML password policy is described at http://www.novell.com/coolsolutions/feature/18589.html.

C: UTF-8 string

pwLen

(OUT) Length of the random password in bytes.

C: size_tint

pwd

(OUT) Random password.

C: UTF-8 string

Return Codes

Returns 0 if successful; not equal to 0 if not successful and an NMAS error code is returned.

See Also

nmasldap_get_user_random_password (page 153)

8 Identification Method Function

NMAS_LOGIN_IDENTITY (page 159) is a new NMAS™ login identity method that replaces NMAS_GetUserName (obsolete 3/03) (page 168) and simplifies the user NMAS GUI interface. The NMAS Identification Method function enables you to invoke user-specified login methods for your Novell® Login Client. This section includes the following topics:

- Section 8.1, “Identification Method Function Descriptions,” on page 155
- Section 8.2, “Login Control Registry Settings,” on page 157
- Section 8.3, “NMAS Log-in Method Function Descriptions,” on page 158

8.1 Identification Method Function Descriptions

When NMAS Login is invoked through Client32™, it can optionally use a third-party DLL to retrieve the username. The DLL must export the API described by NMAS_LOGIN_IDENTITY (page 159). This function enables you to use a wide range of authentication methods including:

- Biometric scanning devices (fingerprint, iris, retina, lips, etc.).
- Token devices (“smart” cards and other physical devices that contain encoded user information).
- Conventional password login methods.

8.1.1 Implementing an ID Plug-in

1. Similar to the device monitor plug-in, implementing an ID plug-in requires compiling a DLL that exports register, start, and stop calls. To do this, the ID plug-in must implement three entry points:
   - *LPFN_NMAS_RegisterIdentityPlugin (page 161)
   - *LPFN_NMAS_StartIdentityPlugin (page 163)
   - *LPFN_NMAS_StopIdentityPlugin (page 164)
2. A fourth entry point, *LPFN_NMAS_SetLoginIdentity (page 162), also is implemented by the NMAS client GUI.

   Just before the NMAS client GUI is displayed, it looks in the registry for the value:

   ```
   IDDLPATH
   ```
3. The DLL specified net registry value then calls the *LPFN_NMAS_RegisterIdentityPlugin (page 161) entry point. When the entry point is called, it passes a pointer to its *LPFN_NMAS_SetLoginIdentity (page 162) call. This function then expects to receive a pointer to the DLL’s *LPFN_NMAS_StartIdentityPlugin (page 163) and *LPFN_NMAS_StopIdentityPlugin (page 164) calls.

When the NMAS client GUI has initialized, it calls the DLL `NMAS_StartIdentityPlugin` through the pointer supplied in the register call. This tells the plug-in to start monitoring its login device to determine the user who corresponds to that device.

4. When the ID plug-in receives the user information, it can then call the *LPFN_NMAS_SetLoginIdentity (page 162) function through the pointer provided in the registry call. The NMAS login identity structure is then passed in, which provides the required identity information.
5. When the NMAS client GUI is ready to unload or when the ID plug-in needs to stop, *LPFN_NMAS_StopIdentityPlugin (page 164) is called. Like the process in secure workstation, after the client GUI makes the StopIdentity call, it might unload the DLL. Consequently, your DLL should ensure that all of its threads are stopped and block *LPFN_NMAS_StopIdentityPlugin (page 164) until the DLL is ready to be unloaded.

   IMPORTANT: The NMAS_StartIdentityPlugin call, like Secure Workstation, can be called multiple times without a call to stop. Consequently, the DLL you compile should be able to detect, for instance, if a thread is monitoring the log-in device and, if so, you must not start another thread.

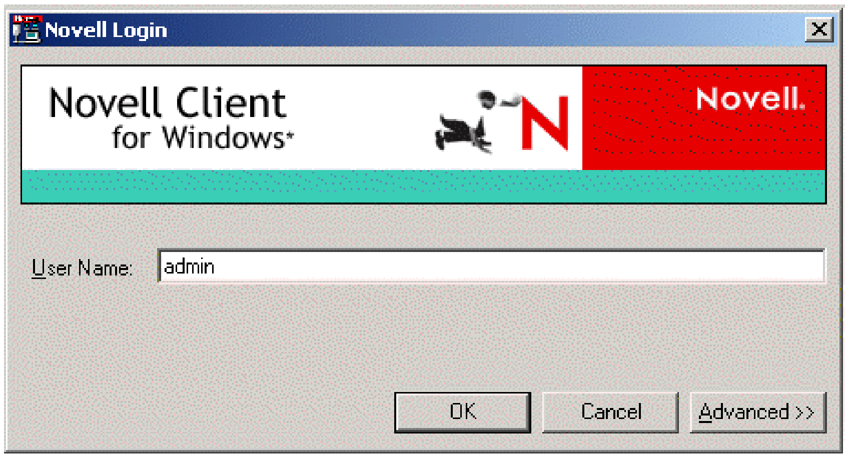
6. The *validFields* variable in NMAS_LOGIN_IDENTITY (page 159) uses the flags described in Valid Flags Define table to provide one field or any combination of the fields contained in this function to the NMAS client GUI. This parameter allows your DLL to identify which fields are provided to the GUI.

   Based on the requirements set by *validFields*, the NMAS client fills either the *userName*, *context*, *tree*, *server*, *sequence*, or *clearance* provided by NMAS_LOGIN_IDENTITY (page 159) into the login dialog box. If configured to do so, the user can then click OK by default, as shown in Figure 8-1 on page 156.

   NOTE: Changing the registry setting changes the default to allow the user to review and change input parameters.

   Figure 8-1 shows an example of an authentication control screen that is returned when a user opts to enter his or her ID manually.

Figure 8-1 *Novell Login Authentication Control Screen*



User Name: admin

8.1.2 Valid Login Method Flags

The *validFields* variable in NMAS_LOGIN_IDENTITY (page 159) can use the flags described below to provide one field or any combination of the fields to the NMAS client GUI. The *validFields* parameter allows your DLL to identify which fields are provided to the GUI:

Valid Flag Defines	Value
NLI_FIELD_USERNAME	0x00000001
NLI_FIELD_CONTEXT	0x00000002
NLI_FIELD_TREE	0x00000004
NLI_FIELD_SERVER	0x00000008
NLI_FIELD_SEQUENCE	0x00000010
NLI_FIELD_CLEARANCE	0x00000020

NOTE: The identity plug-in provides one field or any combination of the fields contained in this function to the NMAS client GUI.

8.2 Login Control Registry Settings

The following registry settings affect the operation of the new login control:

Key	Value	Type	Description
HKLM\Software\Novell\NMAS\1.0\ID	IDDLLPath	String	Contains the full path and name of a DLL that exports the NMAS_GetUserName function.
HKLM\Software\Novell\NMAS\1.0\ID	RestartPluginOnFailure	DWORD	If this value is 1, the client restarts the ID plug-in when the login dialog box is displayed after a failed login. If the value is 0, the client does not restart the ID plug-in after a login failure. The default behavior is 0.
HKLM\Software\Novell\NMAS\1.0\ID	Click *OK*	DWORD	If this value is zero, the NMAS client does not automatically proceed with login after its *LPFN_NMAS_SetLoginIdentity (page 162) function is called. The default behavior is to automatically proceed with login after NMAS_SetLoginIdentity is called.

Key	Value	Type	Description
HKLM\Software\Novell\NMAS\1.0\ID	IDPollInterval (Obsolete 3/2003)	DWORD	This was the “poll interval” for NMAS_GetUserName, but is now obsolete. It specified the time in seconds between calls to NMAS_GetUserName.
HKLM\Software\Novell\NMAS\1.0\ID	IDPrompt	String	Specifies the text displayed in the login dialog box to prompt for an identification device. The default is “Present Card.”
HKLM\Software\Novell\NMAS\1.0\ID	IDWaitPrompt	String	Specifies the text displayed in the login dialog box after the user’s identity has been obtained and the login process has begun. The default is “Please wait...”
HKLM\Software\Novell\NMAS\1.0\ID	ShowCancel	DWORD	If set to 0, the Cancel button in the login dialog box is hidden.
HKLM\Software\Novell\NMAS\1.0\ID	ShowOK	DWORD	If set to 0, the OK button in the login dialog box is hidden.
HKLM\Software\Novell\Login	Advanced	DWROD	If set to 0, the Advanced button in the login dialog box is hidden. This is a Client32 setting, not an NMAS setting.
HKLM\Software\Novell\NMAS\1.0\ID	ShowUsername	DWORD	If set to 0, the Manually Enter Username check box in the login dialog box is hidden.

8.3 NMAS Log-in Method Function Descriptions

The NMAS Log-in Method functions are described below:

- NMAS_LOGIN_IDENTITY (page 159)
- LPFN_DeviceRemoved (page 160)
- *LPFN_NMAS_RegisterIdentityPlugin (page 161)
- *LPFN_NMAS_SetLoginIdentity (page 162)
- *LPFN_NMAS_StartIdentityPlugin (page 163)
- *LPFN_NMAS_StopIdentityPlugin (page 164)
- RegisterDeviceMonitorPlugin (page 165)
- StartDeviceMonitorPlugin (page 166)
- StopDeviceMonitorPlugin (page 167)

NMAS_LOGIN_IDENTITY

Enables the client to pass in all user identification factors. The client DLL can then change what is passed into the buffer.

Syntax

C

```
#include <NMASLoginInfo.h>


struct   NMAS_LOGIN_IDENTITY
(
   int              *size,
   int          *validFields,
   unicode      *userName [NLI_MAX_FIELD_SIZE],
   unicode      *context[NLI_MAX_FIELD_SIZE],
   unicode      *tree[NLI_MAX_FIELD_SIZE],
   unicode      *server[NLI_MAX_FIELD_SIZE],
   unicode      *sequence[NLI_MAX_FIELD_SIZE],
   unicode      *clearance[NLI_MAX_FIELD_SIZE]
);
```

Parameters

`size`

(IN) Set to the size structure of the login device as defined by the parameters in the function.

`validFields`

(IN) Allows your DLL to identify which flag fields are provided to the GUI, using the Valid Login Method Flags.

`userName`

(IN) The user-supplied eDirectory username that is used for login.

`context`

(IN) The context that is used for login.

`tree`

(IN) The tree name that is used for login.

`server`

(IN) The server that is used for login.

`sequence`

(IN) The sequence that is used for login.

`clearance`

(IN) The name of the clearance that is used for login.

LPFN_DeviceRemoved

When the Secure Workstation service calls RegisterDeviceMonitorPlugin (page 165), it provides a pointer to LPFN_DeviceRemoved (page 160). The third-party device monitor plug-in DLL is expected to save the pointer and call this function when the Secure Workstation service detects that its login device has been removed.

Syntax

C

```
#include <wadevmon.h>

typ def   *LPFN_DeviceRemoved
(
   DWORD          sessionid,
   DWORD          methodid
);
```

Parameters

sessionid

(IN) The session ID that was passed to the DLL when StartDeviceMonitorPlugin was called.

methodid

(IN) The NMAS method ID number specified for your login method.

Remarks

For more information about this function, see Section C.4, “Secure Workstation Login Method,” on page 213 and “Implementing A Secure Workstation Plug-in” on page 214.

*LPFN_NMAS_RegisterIdentityPlugin

An identity plug-in must implement this function. The NMAS client login dialog box calls it, passes a pointer to its NMAS_SetLoginIdentity function, and then expects to receive pointers to the NMAS_StartIdentityPlugin and NMAS_StopIdentityPlugin functions in the DLL.

Syntax

C

```
#include <NMASLoginInfo.h>


typedef int   LPFN_NMAS_RegisterIdentityPlugin
(
   LPFN_NMAS_SetLoginIdentity            setIdentity,
   LPFN_NMAS_StartIdentityPlugin        *startIdentityPlugin,
   LPFN_NMAS_StopIdentityPlugin         *stopIdentityPlugin
);
```

Parameters

setIdentity

(IN) Provides a pointer to the NMAS client login dialog's box *LPFN_NMAS_SetLoginIdentity (page 162) function.

startIdentityPlugin

(OUT) Points to the ID plug-in's *LPFN_NMAS_StartIdentityPlugin (page 163) function.

stopIdentityPlugin

(OUT) Points to the ID plug-in's *LPFN_NMAS_StopIdentityPlugin (page 164) function

Remarks

The NMAS client login dialog box calls this function immediately after loading the ID plug-in DLL. This function is called only once each time the NMAS client login dialog box is displayed.

*LPFN_NMAS_SetLoginIdentity

Currently implemented in the NMAS client login dialog box. An identity plug-in DLL should call this function when it has retrieved the user ID.

Syntax

C

```
#include <NMASLoginInfo.h>


int   LPFN_NMAS_SetLoginIdentity
(
   NMAS_LOGIN_IDENTITY              *loginIdentity
);
```

Parameters

`loginIdentity`

(IN) The login information for the user identified by the device.

Remarks

An ID plug-in should call this function when it has retrieved the user ID.

*LPFN_NMAS_StartIdentityPlugin

Used with *LPFN_NMAS_StartIdentityPlugin (page 163), an identity plug-in DLL must implement these calls.

Syntax

C

```
#include <NMASLoginInfo.h>


int   LPFN_NMAS_StartIdentityPlugin ();
```

Remarks

The NMAS client login dialog box calls this entry point to tell the ID plug-in that it should start monitoring its device. After this API has been called, the ID plug-in can call the login dialog *LPFN_NMAS_SetLoginIdentity (page 162) entry point with login information for the user.

The NMAS client login dialog box can make multiple calls to this entry point without calling *LPFN_NMAS_StopIdentityPlugin (page 164). An ID plug-in must detect this condition and handle it accordingly.

*LPFN_NMAS_StopIdentityPlugin

Used with *LPFN_NMAS_StopIdentityPlugin (page 164), an identity plug-in DLL must implement these calls.

Syntax

C

```
#include <NMASLoginInfo.h>

typedef int   LPFN_NMAS_StopIdentityPlugin ();
```

Remarks

The NMAS client login dialog box calls this function to tell an ID plug-in to stop monitoring its device. The ID plug-in DLL might be unloaded after this entry point returns, so the ID plug-in must block in this call until all of its threads have stopped and it is ready to be unloaded.

RegisterDeviceMonitorPlugin

This function is called when the Secure Workstation service starts and your DLL is loaded. Your DLL can call the function pointer to LPFN_DeviceRemoved (page 160) when it detects that its login device has been removed.

Syntax

C

```
#include <wadevmon.h>


int   RegisterDeviceMonitorPlugin
(
   LPFN_DeviceRemoved              DeviceRemoved
);
```

Parameters

`DeviceRemoved`

(IN) Provides a function pointer to LPFN_DeviceRemoved (page 160) (defined in wadevmon.h) that is called by the DLL when it detects that its login device has been removed.

StartDeviceMonitorPlugin

The Secure Workstation service calls this function to tell your DLL to start monitoring its device.

Syntax

C

```
#include <wadevmon.h>

int   StartDeviceMonitorPlugin
(
   DWORD              sessionid
);
```

Parameters

sessionid

(IN) The Windows terminal services session ID, which is usually 0 (unless you're using terminal services or fast user switching).

Remarks

This function can be called when a new user logs on to the workstation (with or without the Secure Workstation PCLM), when the workstation is unlocked, or when a user logs into Client32 using a sequence that includes the Secure Workstation PLCM. The Secure Workstation service can call this function multiple times without calling StopDeviceMonitorPlugin (page 167).

The plug-in must store the session ID provided in this call and pass it to Secure Workstation's LPFN_DeviceRemoved (page 160) call.

StopDeviceMonitorPlugin

The Secure Workstation service calls this function to tell your DLL to stop monitoring its login device. This function can be called when the workstation is locked, or when a user logs out of the workstation.

Syntax

C

```
#include <wadevmon.h>

int   StopDeviceMonitorPlugin
(
   DWORD              sessionid
);
```

Parameters

`sessionid`

(IN) The terminal services session ID, which is passed in from StartDeviceMonitorPlugin (page 166), is currently not used but is provided now to support future implementation of terminal session services.

Remarks

After the Secure Workstation service calls this function, the DLL might be unloaded.

IMPORTANT: Your device monitor plug-in DLL might be unloaded after this call returns. If your DLL spawned a thread in RegisterDeviceMonitorPlugin (page 165) to monitor the log-in device, you must block StartDeviceMonitorPlugin (page 166) until the thread is stopped.

NMAS_GetUserName (obsolete 3/03)

Enables the client to pass in all user identification factors, while DWord allocates the buffer size (512 bytes). However, it is now obsolete and is replaced by NMAS_LOGIN_IDENTITY (page 159). The client DLL can then change what is passed into the buffer. These interfaces are only available on the Client32/Windows platform.

Syntax

C

```
#include <nmas.h>

int   NMAS_GetUserName
(
   unicode        *userName,
   DWORD          *userNameLen,
   unicode        *context,
   DWORD          *contextLen,
   unicode        *tree,
   DWORD          *treeLen,
   unicode        *server,
   DWORD          *serverLen,
   unicode        *sequence,
   DWORD          *sequenceLen,
   unicode        *clearance,
   DWORD          *clearanceLen,
   unicode        *errorMessage,
   DWORD          *errorMessageLen
);
```

Parameters

`userName`

(IN/OUT) The full distinguished name of the user.

`userNameLen`

(IN/OUT) The number of bytes contained in the userName buffer.

`context`

(IN/OUT) IN: The user-supplied eDirectory context. OUT: The eDirectory context that is used for login.

`contextLen`

(IN/OUT) IN: The number of bytes contained in the context buffer. OUT: The length of the string contained in the context buffer.

`tree`

(IN/OUT) IN: User-supplied tree name. OUT: The tree name that is used for login.

`treeLen`

(IN/OUT) IN: The number of bytes in the tree buffer. OUT: The length of the string contained in the context buffer.

`server`

(IN/OUT) IN: User-supplied server name of IP address. OUT: The server that is used for login.

`serverLen`

(IN/OUT) IN: The number of bytes in the server buffer. OUT: The length of the string contained in the server buffer.

`sequence`

(IN/OUT) *IN:* The user-supplied sequence. This is the sequence that the user selects on the NMAS tab. OUT: The sequence that is used for login.

`sequenceLen`

(IN/OUT) IN: The number of bytes in the sequence buffer. OUT: The length of the string contained in the sequence buffer.

`clearance`

(IN/OUT) *IN:* The name of the clearance that the user selected from the NMAS tab. OUT: The name of the clearance that is used for login.

`clearanceLen`

(IN/OUT) IN: The number of bytes in the clearance buffer. OUT: The length of the string contained in the clearance buffer.

`errorMessage`

(OUT) If an error occurs, this parameter should contain a meaningful error message.

`errorMessageLen`

(IN/OUT) IN: The number of bytes in the errorMessage buffer. OUT: The length of the string contained in the errorMessage buffer.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

9 NMAS Javadoc References

To review all of the Javadoc references for the NMAS NDK, see NMAS Javadoc (../api/index.html). The following classes are defined in the NMAS `com.novell.security.nmas.mgmt` package:

9.1 NMAS Interface Classes

- GamsTransport (../api/com/novell/security/nmas/mgmt/GamsTransport.html)
- LoginDataTransport (../api/com/novell/security/nmas/mgmt/LoginDataTransport.html)
- PwdTransport (../api/com/novell/security/nmas/mgmt/PwdTransport.html)

9.2 NMAS Summary Classes

- GamsJLdapTransport (../api/com/novell/security/nmas/mgmt/GamsJLdapTransport.html)
- GamsLdapTransport (../api/com/novell/security/nmas/mgmt/GamsLdapTransport.html)
- GamsMgr (../api/com/novell/security/nmas/mgmt/GamsMgr.html)
- LoginDataJLdapTransport (../api/com/novell/security/nmas/mgmt/LoginDataJLdapTransport.html)
- LoginDataLdapTransport (../api/com/novell/security/nmas/mgmt/LoginDataLdapTransport.html)
- NMASChallengeResponse (../api/com/novell/security/nmas/mgmt/NMASChallengeResponse.html)
- NMASChallengeResponseMgr (../api/com/novell/security/nmas/mgmt/NMASChallengeResponseMgr.html)
- NMASLoginDataMgr (../api/com/novell/security/nmas/mgmt/NMASLoginDataMgr.html)
- NMASPwdMgr (../api/com/novell/security/nmas/mgmt/NMASPwdMgr.html)
- NMASPwdStatus (../api/com/novell/security/nmas/mgmt/NMASPwdStatus.html)
- NMASSimplePwd (../api/com/novell/security/nmas/mgmt/NMASSimplePwd.html)
- NMASSimplePwdMgr (../api/com/novell/security/nmas/mgmt/NMASSimplePwdMgr.html)
- PwdJLdapTransport (../api/com/novell/security/nmas/mgmt/PwdJLdapTransport.html)
- PwdLdapTransport (../api/com/novell/security/nmas/mgmt/PwdLdapTransport.html)

9.3 NMAS Exception Classes

- GamsException (../api/com/novell/security/nmas/mgmt/GamsException.html)

- NMASChallengeResponseException (../api/com/novell/security/nmas/mgmt/NMASChallengeResponseException.html)
- NMASLoginDataException (../api/com/novell/security/nmas/mgmt/NMASLoginDataException.html)
- NMASPwdException (../api/com/novell/security/nmas/mgmt/NMASPwdException.html)

9.4 NMAS Constants

- NMASConstants (../api/com/novell/security/nmas/NMASConstants.html)

A NMAS Error Codes

Decimal Value	Hexadecimal Value	Name	Description
	0	NMAS_SUCCESS	The requested function completed successfully.
-1631	0xFFFFF9A1	NMAS_E_FRAG_FAILURE	Indicates that the NMAS™ NCP™ handler failed.
-1632	0xFFFFF9A0	NMAS_E_BAD_REQUEST_SYNTAX	Indicates that the NMAS NCP handler failed.
-1633	0xFFFFF99F	NMAS_E_BUFFER_OVERFLOW	Buffer passed to MAF_GetAttribute is not large enough to store the requested data.
-1634	0xFFFFF99E	NMAS_E_SYSTEM_RESOURCES	Message returned when NMAS server fails to obtain information from NICI about encryption keys.
-1635	0xFFFFF99D	NMAS_E_INSUFFICIENT_MEMORY	The NMAS server failed to allocate memory for some system use.
-1636	0xFFFFF99C	NMAS_E_NOT_SUPPORTED	Indicates that the login request operation is not supported by the current configuration of NMAS. The error might be returned if the NMAS Client and the NMAS Server versions are not the same. It also might be returned if a login method that does not support the disconnected login is invoked when in the disconnected mode.
-1637	0xFFFFF99B	NMAS_E_BUFFER_UNDERFLOW	The packet size specified in the NMAS packet is larger than the actual size of the packet.
-1638	0xFFFFF99A	NMAS_E_NOT_FOUND	This error might be returned if an object does not exist for the requested username and context.
-1639	0xFFFFF999	NMAS_E_INVALID_OPERATION	Indicates that an NMAS internal error has occurred, caused by the NMAS protocol getting out of order.

Decimal Value	Hexadecimal Value	Name	Description
-1640	0xFFFFF998	NMAS_E_ASN1_DECODE	Failed to decode NMAS internal structures in a signed module.
-1641	0xFFFFF997	NMAS_E_ASN1_ENCODE	Failed to encode NMAS internal structures.
-1642	0xFFFFF996	NMAS_E_LOGIN_FAILED	Indicates that the secret (for example password) presented by the user is invalid.
-1643	0xFFFFF995	NMAS_E_INVALID_PARAMETER	An invalid parameter was passed to the NMAS function.
-1644	0xFFFFF994	NMAS_E_TIMED_OUT_RECOVERABLE	The Client or the server failed to respond in a timely manner. The calling software has the option to retry the request if this error occurs.
-1645	0xFFFFF993	NMAS_E_TIMED_OUT_NOT_ RECOVERABLE	The Client or the server failed to respond in a timely manner. The calling software does not have the option to retry the request if this error occurs.
-1646	0xFFFFF992	NMAS_E_TIMED_OUT_UNKNOWN	The Client or the server failed to respond in a timely manner. The calling software does not have the option to retry the request if this error occurs.
-1647	0xFFFFF991	NMAS_E_AUTH_FAILURE	The creation of eDirectory™ background authentication materials failed.
-1648	0xFFFFF990	NMAS_E_INVALID_DN	Invalid user distinguished name specified for login.
-1649	0xFFFFF98F	NMAS_E_NO_RESOLVE_DN	NMAS was not able to resolve the specified user distinguished name.
-1650	0xFFFFF98E	NMAS_E_NO_RESOLVE_CONN	Not used.
-1651	0xFFFFF98D	NMAS_E_NO_CRYPTO	The Client and server did not negotiate a session key to be used for MAF_XRead, MAF_XWrite, or MAF_XWriteRead.
-1652	0xFFFFF98C	NMAS_E_INVALID_VERSION	1) Indicates that the NMAS Client and NMAS Server versions are incompatible. 2) Indicates that the NMAS method and the NMAS Server versions are incompatible.
-1653	0xFFFFF98B	NMAS_E_SYNC_NEEDED	Indicates there is a problem with the SASDFM key exchange being used.

Decimal Value	Hexadecimal Value	Name	Description
-1654	0xFFFFF98A	NMAS_E_PROTOCOL_STATE	Indicates that the Client or server failed for an unknown reason.
-1655	0xFFFFF989	NMAS_E_INVALID_HANDLE	The NMAS Handle (also known as NMAS_ID) passed to the NMAS function is invalid.
-1656	0xFFFFF988	NMAS_E_INVALID_METHOD	1) NMAS Server failed to validate the signature of a method.
			2) The module size encoded in the signed portion of the method is larger than code read from eDirectory.
-1657	0xFFFFF987	NMAS_E_DEVELOPMENT_VERSION	Used internally by NMAS.
-1658	0xFFFFF986	NMAS_E_MISSING_KEY	The key attribute for the Login Configuration attribute or the Login Secret attribute is missing or corrupt.
-1659	0xFFFFF985	NMAS_E_ACCESS_NOT_ALLOWED	Indicates that the user does not have sufficient rights to perform the requested operation.
-1660	0xFFFFF984	NMAS_E_SEQUENCE_NOT_FOUND	The specified NMAS login sequence is invalid.
-1661	0xFFFFF983	NMAS_E_CLEARANCE_NOT_FOUND	The specified NMAS login clearance is invalid.
-1662	0xFFFFF982	NMAS_E_LSM_NOT_FOUND	An LSM specified in a login sequence is not available for the server platform.
-1663	0xFFFFF981	NMAS_E_LCM_NOT_FOUND	An LCM specified in a login sequence is not available for the Client platform.
-1664	0xFFFFF980	NMAS_E_SERVER_NOT_FOUND	The specified server was not found during NMAS login.
-1665	0xFFFFF97F	NMAS_E_LOGIN_ATTRIBUTE_ NOT_FOUND	The login secret for a particular login method is not available; for example, password not set, fingerprint or biometric data not available.
-1666	0xFFFFF97E	NMAS_E_LEGACY_INVALID_PASSWORD	If a Client doesn't find an NMAS server, it tries the legacy NDS® login method and returns this message if the password fails.
-1667	0xFFFFF97D	NMAS_E_ACCOUNT_DISABLED	A user account has been disabled as a result of intruder detection or administrator action.

Decimal Value	Hexadecimal Value	Name	Description
-1668	0xFFFFF97C	NMAS_E_ACCOUNT_LOCKED	A user account has been locked as a result of intruder detection.
-1669	0xFFFFF97B	NMAS_E_ADDRESS_RESTRICTION	Violation of approved login addresses that are registered on the user object.
-1670	0xFFFFF97A	NMAS_E_CONN_CLEARED	Indicates loss of connection to the server.
-1671	0xFFFFF979	NMAS_E_TIME_RESTRICTION	Violations of time restrictions that are set on the user object for logging in to a server.
-1672	0xFFFFF978	NMAS_E_SHORT_TERM_SECRET	Not used. Replaced by NMAS_E_AUTH_FAILED.
-1673	0xFFFFF977	NMAS_E_NO_NMAS_ON_TREE	NMAS is not installed in the specified tree. This typically occurs when using the method management API.
-1674	0xFFFFF976	NMAS_E_NO_NMAS_ON_SERVER	NMAS is not installed on the specified server. This typically occurs when using the method management API.
-1675	0xFFFFF975	NMAS_E_REQUEST_CHALLENGED	The normal error code returned from a proxy LCM to indicate that a challenge was requested by the LSM.
-1676	0xFFFFF974	NMAS_E_LOGIN_CANCELED	The normal error code returned to NMAS from a method if the user cancelled the login from a LCM.
-1677	0xFFFFF973	NMAS_E_LOCAL_CRED_STORE	Not used.
-1678	0xFFFFF972	NMAS_E_REMOTE_CRED_STORE	Not used.
-1679	0xFFFFF971	NMAS_E_SMC_NICM	Not used.
-1680	0xFFFFF970	NMAS_E_SEQUENCE_NOT_ AUTHORIZED	Although a login sequence is valid, the requested user is not authorized to use it to log in.
-1681	0xFFFFF96F	NMAS_E_TRANSPORT	A transport callback routine was not provided to the NMAS Client.
-1682	0xFFFFF96E	NMAS_E_CRYPTO_FAILED_INIT	A rare cryptography initialization error that can occur because of a failure in the framework of the host computer.
-1683	0xFFFFF96D	NMAS_E_DOUBLEBYTE_FAILED_INIT	A rare initialization error that can occur because of a failure in the framework of the host computer.

Decimal Value	Hexadecimal Value	Name	Description
-1684	0xFFFFF96C	NMAS_E_CODEPAGE_FAILED_INIT	A rare initialization error that can occur because of a failure in the framework of the host computer.
-1685	0xFFFFF96B	NMAS_E_UNICODE_FAILED_INIT	A rare initialization error that can occur because of a failure in the framework of the host computer.
-1686	0xFFFFF96A	NMAS_E_DLL_FAILED_LOADING	NMAS Client failed to load.
-1687	0xFFFFF969	NMAS_E_EVAL_VERSION_WARNING	Indicates that the NMAS Evaluation Edition is being used to log in.
-1688	0xFFFFF968	NMAS_E_CONCURRENT_LOGIN	Violation of the number of workstations that a user can log in to concurrently, as assigned by the network administrator.
-1689	0xFFFFF967	NMAS_E_THREAD_CREATE	An internal error seen primarily on NetWare 6 when NMAS fails to create a thread.
-1690	0xFFFFF966	NMAS_E_SECURE_CHANNEL_ REQUIRED	SSL was not used when attempting to administer NMAS through the NMAS LDAP extension functions.
-1691	0xFFFFF965	NMAS_E_NO_DEFAULT_USER_ SEQUENCE	A login sequence was not specified during login and the user does not have a default login sequence.
-1692	0xFFFFF964	NMAS_E_NO_TREENAME	NMAS was unable to get the tree name from eDirectory.
-1693	0xFFFFF963	NMAS_E_MECH_NOT_FOUND	The specified Simple Authentication Security Layer (SASL) mechanism is not available.
-1694	0xFFFFF962	NMAS_E_ACCOUNT_NOT_ACTIVATED	The account has been created but the date and time that the account can be activated has not been reached.
-1695	0xFFFFF961	NMAS_E_INCOMPATIBLE_LOGIN_DATA	The stored login data cannot be used by the login method to validate the user. For example, if the password is stored as a digest value, such as SHA-1 and the DIGEST-MD5 login method is used to log in, the login fails with this error
-1696	0xFFFFF960	NMAS_E_PASSWORD_HISTORY_FULL	The password change failed because the password history for the user cannot store any more passwords.

Decimal Value	Hexadecimal Value	Name	Description
-1697	0xFFFFF95F	NMAS_E_INVALID_SPM_REQUEST	The requested password operation is invalid.
-1698	0xFFFFF95E	NMAS_E_PASSWORD_MISMATCH	The password change failed because the old password provided by the user does not match the user's current password.
-1699	0xFFFFF95D	NMAS_E_OBSOLETE_METHOD	An attempt to update the login method failed because the current version is newer.
-16000	0xFFFFC180	NMAS_E_PASSWORD_TOO_LONG	The password change or set request failed because the password is longer than allowed by the password policy.
-16001	0xFFFFC17F	NMAS_E_PASSWORD_UPPER_MIN	The password change or set request failed because the password does not contain the minimum number of uppercase characters required by the password policy.
-16002	0xFFFFC17E	NMAS_E_PASSWORD_UPPER_MAX	The password change or set request failed because the password contains more than the maximum number of uppercase characters allowed by the password policy.
-16003	0xFFFFC17D	NMAS_E_PASSWORD_LOWER_MIN	The password change or set request failed because the password does not contain the minimum number of lowercase characters required by the password policy.
-16004	0xFFFFC17C	NMAS_E_PASSWORD_LOWER_MAX	The password change or set request failed because the password contains more than the maximum number of lowercase characters allowed by the password policy.
-16005	0xFFFFC17B	NMAS_E_PASSWORD_NUMERIC_DISALLOWED	The password change or set request failed because the password contains numeric characters that are disallowed by the password policy.
-16006	0xFFFFC17A	NMAS_E_PASSWORD_NUMERIC_FIRST	The password change or set request failed because the first character of the password is a numeric character that is disallowed by the password policy.

Decimal Value	Hexadecimal Value	Name	Description
-16007	0xFFFFC179	NMAS_E_PASSWORD_NUMERIC_LAST	The password change or set request failed because the last character of the password is a numeric character that is disallowed by the password policy.
-16008	0xFFFFC178	NMAS_E_PASSWORD_NUMERIC_MIN	The password change or set request failed because the password does not contain the minimum number of numeric characters required by the password policy.
-16009	0xFFFFC177	NMAS_E_PASSWORD_NUMERIC_MAX	The password change or set request failed because the password contains more than the maximum number of numeric case characters allowed by the password policy.
-16010	0xFFFFC176	NMAS_E_PASSWORD_SPECIAL_ DISALLOWED	The password change or set request failed because the password contains non-alphanumeric characters that are disallowed by the password policy.
-16011	0xFFFFC175	NMAS_E_PASSWORD_SPECIAL_FIRST	The password change or set request failed because the first character of the password is a non-alphanumeric character that is disallowed by the password policy.
-16012	0xFFFFC174	NMAS_E_PASSWORD_SPECIAL_LAST	The password change or set request failed because the last character of the password is a non-alphanumeric character that is disallowed by the password policy.
-16013	0xFFFFC173	NMAS_E_PASSWORD_SPECIAL_MIN	The password change or set request failed because the password does not contain the minimum number of non-alphanumeric characters required by the password policy.
-16014	0xFFFFC172	NMAS_E_PASSWORD_SPECIAL_MAX	The password change or set request failed because the password contains more than the maximum number of non-alphanumeric case characters allowed by the password policy.
-16015	0xFFFFC171	NMAS_E_PASSWORD_REPEAT_ CHAR_MAX	The password change or set request failed because the password contains a character that appears in the password more times than is allowed by the password policy.

Decimal Value	Hexadecimal Value	Name	Description
-16016	0xFFFFC170	NMAS_E_PASSWORD_CONSECUTIVE_MAX	The password change or set request failed because the password contains a character that appears consecutively more times than is allowed by the password policy.
-16017	0xFFFFC16F	NMAS_E_PASSWORD_UNIQUE_MIN	The password change or set request failed because the password does not contain the minimum number of unique characters required by the password policy.
-16018	0xFFFFC16E	NMAS_E_PASSWORD_LIFE_MIN	The password change or set request failed because the minimum amount of time since the last successful password change as required by the password policy has not elapsed.
-16019	0xFFFFC16D	NMAS_E_PASSWORD_EXCLUDE	The password change or set request failed because the password appears in the excluded password list of the password policy.
-16020	0xFFFFC16C	NMAS_E_PASSWORD_ATTR_VALUE	The password change or set request failed because the password is the same as a value of a user attribute that is disallowed by the password policy.
-16021	0xFFFFC16B	NMAS_E_PASSWORD_EXTENDED_DISALLOWED	The password change or set request failed because the password contains extended characters that are disallowed by the password policy.
-16022	0xFFFFC16A	NMAS_E_INVALID_PASSWORD_POLCY	The password policy associated with the user is not a valid password policy.
-16023	0xFFFFC169	NMAS_E_LOGIN_FAILED_ERROR_DISPLAYED	The password change or set request failed because NMAS experienced an internal error.
-16024	0xFFFFC168	NMAS_E_CFG_METHOD_EXISTS	The client configuration already contains the method data. The result of trying to add a duplicate method to the configuration data.
-16025	0xFFFFC167	NMAS_E_CFG_NO_DATA	No client configuration data exists.
-16026	0xFFFFC166	NMAS_E_CFG_NO_ACCESS	No access to client configuration data.

Decimal Value	Hexadecimal Value	Name	Description
-16027	0xFFFFC165	NMAS_E_CFG_NO_MORE_ITEMS	No more items in the client configuration data. Signals the end of the data enumeration.
-16028	0xFFFFC164	NMAS_E_CFG_METHOD_NOT_FOUND	The client configuration data does not contain the method information.
-16029	0xFFFFC163	NMAS_E_CFG_INVALID_DATA	The client configuration data is invalid.
-16030	0xFFFFC162	NMAS_E_CFG_IDPLUGIN_EXISTS	The client configuration data already contains a identity plugin entry.
-16031	0xFFFFC161	NMAS_E_CFG_IDPLUGIN_NOT_FOUND	No identity plugin data was found in the client configuration.
-16032	0xFFFFC16	NMAS_E_CFG_TRACEINFO_EXISTS	The trace information exits in the configuration data.
-16033	0xFFFFC15F	NMAS_E_CFG_TRACEINFO_NOT_FOUND	The trace information was not found in the configuration data.
-16034	0xFFFFC15E	NMAS_E_PASSWORD_UPPER_FIRST	The first character of the password does not allow uppercase characters.
-16035	0xFFFFC15C	NMAS_E_PASSWORD_UPPER_LAST	The last character of the password does not allow uppercase characters.
-16036	0xFFFFC169	NMAS_E_LOGIN_FAILED_ERROR_DISPLAYED	Displays “NMAS internal error.”
-16037	0xFFFFC15B	NMAS_E_PASSWORD_LOWER_LAST	The last character of the password does not allow lowercase characters.
-16038	0xFFFFC15A	NMAS_E_PASSWORD_EXTENDED_FIRST	The first character of the password does not allow extended characters.
-16039	0xFFFFC159	NMAS_E_PASSWORD_EXTENDED_LAST	The last character of the password does not allow extended characters.
-16040	0xFFFFC158	NMAS_E_PASSWORD_EXTENDED_MIN	The password does not contain the minimum number of extended characters.
-16041	0xFFFFC157	NMAS_E_PASSWORD_EXTENDED_MAX	Password contains more than the maximum number of extended characters.
-16042	0xFFFFC156	NMAS_E_PASSWORD_UPPER_DISALLOWED	The password does not allow uppercase characters.
-16043	0xFFFFC155	NMAS_E_PASSWORD_LOWER_DISALLOWED	The password does not allow lowercase characters.

Decimal Value	Hexadecimal Value	Name	Description
-16044	0xFFFFC154	NMAS_E_USER_CHALLENGES_NOT_SYNCED	User challenge questions do not match the challenge questions in the challenge set associated with the user.
-16045	0xFFFFC153	NMAS_E_INVALID_USER_CHALLENGE_SET	The user challenge questions are not formatted properly.
-16046	0xFFFFC152	NMAS_E_CFG_INFO_EXISTS	***
-16047	0xFFFFC151	NMAS_E_CFG_INFO_NOT_FOUND	***
-16048	0xFFFFC150	NMAS_E_ENTRY_EXISTS	This may be returned from MAFDS_ModifyEntry (page 100). It indicates that the user either 1) tried to add a duplicate value to a multiple-valued attribute or 2) tried to add multiple values to single valued attribute.
-16049	0xFFFFC14F	NMAS_E_ENTRY_ATTRIBUTE_NOT_FOUND	The requested attribute does not exist on the specified object.
-16050	0xFFFFC14E	NMAS_E_NO_MORE_ENTRY_ATTRIBUTES	All of the attributes values have been returned to the calling routine.
-16051	0xFFFFC14D	NMAS_E_UNKNOWN_IDENTITY	The specified connection does not have a user identity.
-16052	0xFFFFC14C	NMAS_E_ENTRY_NOT_FOUND	The specified object does not exist.
-16053	0xFFFFC14B	NMAS_E_ATTRIBUTE_EXISTS	The object already has the specified attribute value or the attribute can not have multiple values.
-16054	0xFFFFC14A	NMAS_E_AUDIT_REQUIRED	NMAS Auditing is required but is not configured or installed correctly.
-16055	0xFFFFC149	NMAS_E_PASSWORD_AD2K8_COMPLEX_VIOLATION	The password change or set request failed because the password violates the complexity rules of the assigned Microsoft Server 2008-format password policy.
-16056	0xFFFFC148	NMAS_E_PASSWORD_RANDOM_FAILED	The password change or set request failed because the NMAS server could not generate a random password conforming to the rules of the assigned password policy.

Decimal Value	Hexadecimal Value	Name	Description
-16057	0xFFFFC147	NMAS_E_PASSWORD_NONALPHA_DISALLOWED	The password change or set request failed because the password contains a non-alphabetic character. The assigned password policy does not allow non-alphabetic characters.
-16058	0xFFFFC146	NMAS_E_PASSWORD_NONALPHA_MIN	The password change or set request failed because the password does not contain the minimum number of non-alphabetic characters required by the assigned password policy.
-16059	0xFFFFC145	NMAS_E_PASSWORD_NONALPHA_MAX	The password change or set request failed because the password contains more than the maximum number of non-alphabetic characters allowed by the assigned password policy.

B Installing Novell eDirectory

When you extend NMAS™ functionality on your applications, you should already have Novell® eDirectory™ installed. If you do not already have Novell Linux Services (NLS) or eDirectory on your Linux system, you can install it manually.

1 Download the latest version of eDirectory from eDirectory 8.8 SP5 Platform-Specific Downloads (http://download.novell.com/Download?buildid=Um8b-a_q0-g~).

   1a Select the version of eDirectory you wish to install.

   IMPORTANT: During this step, you must log in to your Novell account, which allows you to access all of the secure Novell applications or databases to which you have entitlement rights. If you do not have an account, create a new Novell Account here (https://secure-www.novell.com/selfreg/jsp/createAccount.jsp).

   1b For detailed installation assistance, see Novell eDirectory 8.8 (http://www.novell.com/documentation/edir88/).

2 At the Linux command prompt, uncompress eDirectory by entering

```
tar -zxvf eDirectory_88SP5_Linux_<architecture type>.tar.gz
```

3 Change directories to install the directory by entering

```
cd Linux/setup./nds-install -c serverndsconfig new -t corp-tree -n o=novell-a
cn=admin.o=novell
```

IMPORTANT: To complete the installation, you need the license (.nfk) file.

4 Create the required NMAS objects in eDirectory by entering

```
nmasinst -i admin.novell corp-tree
```

5 Install ConsoleOne® by entering

```
cd ../ConsoleOne > ./c1-install
```

C Deprecated NMAS Functions

This section contains the following deprecated topics and functions that are no longer supported by Novell®:

- Section C.1, "Client Application Login Functions," on page 187
- Section C.2, "NMAS Transport API," on page 190
- Section C.3, "Proxy Functions," on page 196
- Section C.4, "Secure Workstation Login Method," on page 213

C.1 Client Application Login Functions

The NMAS™ client application login functions were used by Windows applications to invoke the NMAS client to log in or log in again. These functions provided a method to force users to log in or log in again using a prescribed login sequence when starting an application:

- NMAS_DisconnectedLogin (page 188)
- NMAS_LegacyRelogin (page 189)

NMAS_DisconnectedLogin provided the ability to force a user to login using NMAS rather than some other authentication method when disconnected. NMAS_LegacyRelogin used the default login sequence of an application.

For example, when an application is using Single Sign-on to access a user's Novell SecretStore™, NMAS_LegacyRelogin (page 189) or NMAS_DisconnectedLogin (page 188) could be used to force a user to log in again before the secrets (that is, ID, password, certificates, etc.) are extracted from Secret Store and presented to the application. In essence, this provides a secondary login procedure to enhance security when authenticating to a network or an application.

NOTE: The NMAS_DisconnectedLogin function is obsolete and is replaced with enhanced disconnected login functionality included with the current NMAS NDK.

NMAS_DisconnectedLogin

Starts the NMAS disconnected login process using the input login and user information.

Syntax

C

```
#include <sasflegy.h>
```

```
nint   NMAS_DisconnectedLogin
(
   pnstr     treeName
   pnstr     subjectDN,
   pnvint8   preGatheredSecrets,
   pnstr     reserved
);
```

Parameters

treeName

(IN) This local code page string specifies the tree to be used for disconnected login.

subjectDN

(IN) This local code page string specifies the full distinguished name (DN) of the user to be used for disconnected login. The DN must be dot-delimited and typeless.

preGatheredSecrets

(IN) Optional password.

reserved

(IN) Must be null.

Remarks

This function is intended to be called by client applications to require a user to reauthenticate when not connected to the eDirectory™ tree before the application is available to the user. No attempt is made to access the network when calling this function. This function will become obsolete when new enhanced disconnected login functionality is introduced in the future.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

NMAS_LegacyRelogin

Starts the NMAS login process using the input login and user information or the login and user information used from previous logins.

Syntax

C

```
#include <sasflegy.h>

int   NMAS_LegacyRelogin
(
   pnstr    treeName
   pnstr    subjectDN,
   pnstr    requestedSequence,
   pnstr    reserved1,
   pnstr    reserved2
);
```

Parameters

`treeName`

(IN) This local code page string specifies the tree to be used to logged in again. If this parameter is `null` and/or the `subjectDN` is `null,` the tree and user of the primary login session is used.

`subjectDN`

(IN) This local code page string specifies the full distinguished name (DN) of the user to be logged in again. The DN must be dot-delimited and typeless. If this parameter is `null` and/or the `treeName` is `null,` the tree and user of the primary login session is used.

`requestedSequence`

(IN) This local code page string specifies the login sequence to be used to logged in again. If this parameter is null, this function uses the login sequence that was used for the first login or the user's default login sequence.

`reserved1`

Must be null.

`reserved2`

Must be null.

Remarks

This function is intended to be called by client applications to require a user log in again before the application is available to the user. No new authenticated login sessions result from calling this function. Only available with the Novell® Client32™.

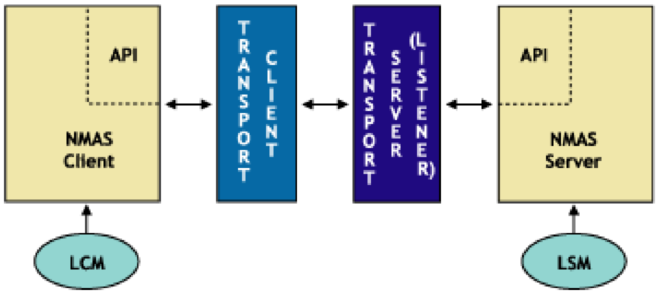
Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

C.2 NMAS Transport API

The diagram in Figure C-1 represents how the NMAS Transport API interacts on the client and server, respectively. The NMAS Transport API provides a means to invoke NMAS to perform a login using the transport provided by the caller.

Figure C-1 *High-Level Architecture of The NMAS Transport API*



For example, LDAP can use NMAS to perform the login by invoking the NMAS Client and supplying a transport callback routine and handle. The NMAS Client calls the transport callback to send data to the server. The transport handler ("listener") on the Server calls an NMAS Server with the data sent by the NMAS Client.

NMAS_ClientLogin

Called by the client transport to initiate the NMAS login.

Syntax

C

```
#include <nmasTransport.h>

int   NMAS_ClientLogin
(
   NMAS_ClientLoginRequestInfo    *rInfo,
   NMAS_ClientLoginTransportFcn    transportFcn,
   int                             transportHandle,
   long                            features,
   long                            options,
   void                           *reserved1,
   void                            *reserved2
);
```

Parameters

`rInfo`

(IN) This is a structure that contains the user login request information (for example, user distinguished name). See types NMAS_ClientLoginRequestInfo (page 192).

`transportFcn`

(IN) This is the transport call back routine which is used by the NMAS client to send data to the server. See types NMAS_ClientLoginTransportFcn (page 192).

`transportHandle`

(IN) This is the transport-specific handle which is passed by the NMAS client as the first parameter of the call back routine.

`features`

This is a flags parameter which is intended to be used to request login features. No feature flags currently have been defined. Currently only defined feature flag specifies that the NMAS client is disconnected from the NMAS server. The value for this feature is NMAS_TRANSPORT_DISCONNECTED (8).

`options`

This flags parameter is intended to specify the behavior of the NMAS login session. Currently the only defined option flag specifies that NMAS not perform any encryption of data that is transported between the NMAS client and NMAS server. The value for this option flag is NMAS_TRANS_PORT_OPTION_NO_CRYPTO (8).

`reserved1`

Must be null.

`reserved2`

Must be null.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_MessageHandler (page 194)

NMAS_ClientLoginRequestInfo

Contains the user's login request information that is passed to NMAS_ClientLogin (page 191).

```
#include <nmasTransport.h>

struct    NMAS_ClientLoginRequestInfo
{
   unicode              *userName;
   unicode              *reqSequence;
   unicode              *treeName;
   unicode              *reqClearance;
   unicode              *password
;}
```

userName

The full distinguished name (typeless dot-delimited) of the user.

reqSequence

(Optional) The optional login sequence field that is requested by the user.

treeName

(Optional) The target NDS tree of the login session. This field is optional.

reqClearance

(Optional) The Unicode representation of the session clearance requested by the user. This field is optional.

password

(Optional) The Unicode representation of the user's password. This field is optional.

NMAS_ClientLoginTransportFcn

This is the definition of the transport callback routine which is passed to NMAS_ClientLogin (page 191).

```
#include <nmasTransport.h>

int     NMAS_ClientLoginTransportFcn
(
   int                   handle,
   void                 *instructionBuf,
   size_t                instructionLen,
   void                 *resultBuf,
   size_t                *resultLen;)
```

handle

(IN) The transport specific handle which is passed to NMAS_PutLoginConfig (page 115).

`instructionBuf`

(IN) Contains the request that is to be sent to the NMAS Server.

`instructionLen`

(IN) The number of bytes contained in `instructionBuf`.

`resultBuf`

(OUT) Contains the reply from the NMAS Server. Buffer must be large enough for any reply. Needs a recommended size.

`resultLen`

The number of bytes contained in `resultBuf`.

NMAS_MessageHandler

This function is called by the server side of the transport to process the NMAS client request and to return the NMAS server reply.

Syntax

C

```
#include <nmasMsg.h>


NMASAPI    NMAS_MessageHandler
(
   size_t        inSize,
   char         *inBuffer,
   size_t       *outSize,
   char        **outBuffer,
   nint32       *nmasHandle,
   nuint32      *status
);
```

Parameters

inSize

(IN) The number of bytes contained in inBuffer which was received from the NMAS Client.

inBuffer

(IN) contains the request received from the NMAS Client.

outSize

(OUT) The number of bytes contained in *outBuffer*.

outBuffer

(OUT) Contains the reply to be sent back to NMAS Client. Must be freed by calling NMAS_FreeReply (page 196) after data is sent to client.

nmasID

(OUT) The NMAS identifier which can be used by the transport to make calls to other NMAS functions such as NMAS_Authenticate (page 198) or NMAS_LocalAuthenticate (page 209).

status

(OUT) Indicates if the NMAS login session is continuing or finished.

The following list shows the flag definitions:

FLAG	VALUE
NMAS_TRANS_STATUS_CONT	0x0
NMAS_TRANS_STATUS_SUCCESS	0x1
NMAS_TRANS_STATUS_FAILED	0x2
NMAS_TRANS_STATUS_DONE	0x4
NMAS_TRANS_STATUS_FIRST	0x8

Return Values

A non-zero value indicates that there was a fatal system error which caused the login session to abort.

See Also

NMAS_FreeReply (page 196)
NMAS_Authenticate (page 198)
NMAS_LocalAuthenticate (page 209)

NMAS_FreeReply

Called by the server side of the transport to free the reply buffer which is allocated and returned to the caller of NMAS_MessageHandler (page 194).

Syntax

C

```
#include <nmasMsg.h>

int   NMAS_FreeReply
(
    void       *buffer
);
```

Parameters

buffer

(IN) The reply buffer which was allocated and returned by NMAS_MessageHandler (page 194).

Return Codes

None.

See Also

NMAS_MessageHandler (page 194)

C.3 Proxy Functions

The NMAS Proxy functions allow other services to use NMAS for authentication. The proxy client must meet the minimum requirements to authenticate: it must provide a login sequence, subject name, and a list of available login methods. NMAS_PutAttribute (page 211) is used to specify the login sequence and user name, as well as other information that might be needed by a specific login method.

The proxy client performs the MAF protocol using the NMAS_CanDo (page 200), NMAS_InvokeMethod (page 208), and NMAS_WhatNext (page 212) functions.

When all methods have been invoked and have successfully verified credentials, use NMAS_Authenticate (page 198) or NMAS_LocalAuthenticate (page 209) to create an authenticated connection. This connection must be closed later using NMAS_Logout (page 210).

The following sample shows how to authenticate using the Proxy API set:

```
ProxyLogin()
{
 int nmasID, mafHandle, int ndsContext=INVALID_NDS_CONTEXT;
 MethodIDsBuffer clientCanDo[2];
 MethodIDsBuffer doMethod;

 NWDSCreateContextHandle(&ndsContext);

// Create an NMAS handle. This context must be destroyed later// using
NMAS_DestroyContext.
NMAS_CreateContext(&nmasID);// Save the DN of the user to be logged
inNMAS_PutAttribute(nmasID, NMAS_AID_USERNAME, )// Save the name of the login
sequence
NMAS_PutAttribute(nmasID, NMAS_AID_REQUESTED_METHODS, );// Build an array of login
methods executed by clientmemset(clientCanDo, 0, sizeof(MethodIDsBuffer * 2));
clientCanDo[0].methodLen = 4;
clientCanDo[0].methodID[0] = 0x0000001;

// Login method ID number
clientCanDo[1].methodLen = 4;
clientCanDo[1].methodID[0] = 0x0000009;
// Login method ID number// Get the first method to invoke
NMAS_CanDo(nmasID,
1,           // Number of methods in the clientCanDo array
clientCanDo,0, NULL,     // Reserved
&doMethod,   // Receives the first login method to invoke
&mafHandle); // Receives a MAF handlewhile (doMethod.methodID[0])
{     // Invoke the LCM and LSM for the method.
     NMAS_InvokeMethod(mafHandle, &doMethod);

     // Get the next method to invoke
     NMAS_WhatNext(nmasID, &doMethod, &mafHandle);
}// Create a connection. This connection must be closed later
// using NMAS_Logout.
NMAS_Authenticate(nmasID, mafHandle, INVALID_NDS_CONTEXT, ndsContext);
}
```

NMAS_Authenticate

Authenticates the specified NDS eDirectory context. Call this routine after NMAS_MessageHandler (page 194) returns a SUCCESS status and before a DONE status.

Syntax

```
#include <nmaspxy.h>

nint32   NMAS_Authenticate
(
   nint32   nmasHandle,
   nint32   mafHandle,
   nint     reserved,
   nint     ndsContext
);
```

Parameters

nmasHandle

(IN) The NMAS handle.

mafHandle

(IN) The MAF handle.

reserved

Must be INVALID.NDS.CONTEXT (-1).

ndsContext

(IN) Provides a DSAPI context.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_LocalAuthenticate (page 209)
NMAS_Logout (page 210)

NMAS_AuthenticateConnection

Associates the input connection number (*connID*) with the identity logged in with the input *nmasHandle*. Call this routine after NMAS_MessageHandler (page 194) returns a SUCCESS status and before a DONE status.

Syntax

```
#include <nmaspxy.h>

nint32   NMAS_AuthenticateConnection
(
   nint32   nmasHandle,
   int      connID
);
```

Parameters

`nmasHandle`

(IN) The NMAS handle.

`connID`

(IN) The eDirectory or Netware connection.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_Authenticate (page 198)
NMAS_LocalAuthenticate (page 209)

NMAS_CanDo

Specifies the login methods the proxy client can perform. Returns the first method to be invoked. Invokes the LSM of the first method.

Syntax

```
#include <nmaspxy.h>

 nint32   NMAS_CanDo
(
    nint32             nmasHandle,
    nint32             numMethods,
    MethodIDsBuffer *canDoMethods,
    size_t             reserved1,
    void               reserved2,
    MethodIDsBuffer *doMethod,
    nint32            *mafHandle
);
```

Parameters

nmasHandle

(IN) The NMAS handle.

numMethods

(IN) The number of entries in the *canDoMethods* buffer.

canDoMethods

(IN) Array of Method IDs that the proxy client can do.

reserved1

Reserved. Must be null.

reserved2

Reserved. Must be null.

doMethod

(OUT) Pointer to a *MethodIDsBuffer* structure. Upon success, this parameter contains the Method ID of the login method to be invoked.

mafHandle

(OUT) The MAF handle for use with NMAS_InvokeMethod (page 208) and NMAS_WhatNext (page 212).

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_PutAttribute (page 211)
NMAS_InvokeMethod (page 208)

NMAS_WhatNext (page 212)

NMAS_CreateContext

Creates an NMAS context handle.

Syntax

```
#include <nmaspxy.h>

 nint32   NMAS_CreateContext
(
    nint32   *nmasHandle
);
```

Parameters

`nmasHandle`

(OUT) Receives a pointer to the NMAS context handle.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_DestroyContext (page 203)

NMAS_DestroyContext

Destroys an NMAS context handle.

Syntax

```
#include <nmaspxy.h>

nint32   NMAS_DestroyContext
(
   nint32   nmasHandle
);
```

Parameters

`nmasHandle`

(IN) The NMAS handle to be destroyed.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_CreateContext (page 202)

NMAS_FindMethods

Returns the ordered list of Method IDs that define the input login sequence.

Syntax

```
#include <nmaspxy.h>

nint32   NMAS_FindMethods
(
   unicode          *seqName,
   nint32           *bufferLen,
   MethodIDsBuffer *methods,
   nint32           *methodCnt
);
```

Parameters

`seqName`

(IN) Name of a login sequence.

`bufferLen`

(IN) Size in bytes of the *methods*.

`methods`

(IN/OUT) Ordered list of methods that defines the login sequence.

`methodCnt`

(OUT) Number of Method IDs that are contained in *methods*.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_GetAvailableMethods (page 207)
NMAS_FindLoginSequences (page 205)

NMAS_FindLoginSequences

Iteratively returns the login sequence names that can be used with the specified list of login methods.

Syntax

```
#include <nmaspxy.h>

nint32   NMAS_FindLoginSequences
(
   nint32            nmasHandle,
   nint32            *iter,
   nint32            numMethods,
   MethodIDsBuffer   *methods,
   nint32            bufferLenInBytes,
   void              *seqName
);
```

Parameters

`nmasHandle`

(IN) The NMAS handle.

`iter`

(IN/OUT) On the first iteration, the input value must equal -1 The output value is -1 when no more login sequences can be found. (See Remarks.)

`numMethods`

(IN) The number of entries in the *Methods* buffer.

`methods`

(IN) List of desired methods.

`bufferLenInBytes`

(IN) Length in bytes of sequence name.

seqName

(IN/OUT) Name of login sequence.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

Remarks

Only the names of the login sequences that contain a subset of the input list of methods is returned. To determine list of methods in a login sequence use the NMAS_FindMethods function.

See Also

NMAS_FindMethods (page 204)
NMAS_GetAvailableMethods (page 207)

NMAS_GetAttribute

Retrieves data placed in the attribute store using NMAS_PutAttribute (page 211).

Syntax

```
#include <nmaspxy.h>

 nint32   NMAS_GetAttribute
(
    nint32   nmasHandle,
    nint32   aid,
    void    *reserved,
    nint32   attributeLenInBytes,
    void    *attributeValue
);
```

Parameters

`nmasHandle`

(IN) The NMAS handle.

`aid`

(IN) The type of attribute to be retrieved. See Section 4.2, “NMAS Attribute IDs,” on page 47 for a listing of valid values.

`reserved`

Reserved. Must be null.

`attributeLenInBytes`

(OUT) The length of the buffer pointed to by *attributeValue*.

`attributeValue`

(OUT) The retrieved value.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_PutAttribute (page 211)

NMAS_GetAvailableMethods

Provides a list of proxy Login Client Modules (LCMs) that are available on the server.

Syntax

```
#include <nmaspxy.h>

nint32   NMAS_GetAvailableMethods
(
   nint32              *bufferLen,
   MethodIDsBuffer     *methods,
   nint32              *methodCnt
);
```

Parameters

`bufferLen`

(IN) Size in bytes of the methods.

`methods`

(IN/OUT) Ordered list of Method IDs of proxy LCMs available on the server.

`methodCnt`

(OUT) Number of Method IDs in methods.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_FindMethods (page 204)
NMAS_FindLoginSequences (page 205)

NMAS_InvokeMethod

Executes a login method. The proxy LCM for the selected method is invoked and this call blocks until the login method is complete. A call to this function should be followed by a call to NMAS_WhatNext (page 212).

Syntax

```
#include <nmaspxy.h>

nint32   NMAS_InvokeMethod
(
   nint32              mafHandle,
   MethodIDsBuffer    *doMethod
);
```

Parameters

mafHandle

(IN) The MAF handle returned by *NMAS_CanDo* or NMAS_WhatNext.

doMethod

(IN) The login method to be invoked, returned by *NMAS_CanDo* or NMAS_WhatNext.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

Remarks

The method's Login Server Module is invoked by NMAS_CanDo (page 200) or NMAS_WhatNext (page 212).

See Also

NMAS_CanDo (page 200)
NMAS_WhatNext (page 212)

NMAS_LocalAuthenticate

Allocates and authenticates an NCP connection for use on the local system only. The connection is set as the connection for the specified NDS eDirectory context. The connection must be closed with NMAS_Logout. Call this routine after NMAS_MessageHandler (page 194) returns a SUCCESS status and before a DONE status.

Syntax

```
#include <nmaspxy.h>

nint32   NMAS_LocalAuthenticate
(
   nint32   nmasHandle,
   nint32   mafHandle,
   nint     reserved,
   nint     ndsContext
);
```

Parameters

`nmasHandle`

(IN) The NMAS handle.

`mafHandle`

(IN) The MAF handle.

`reserved`

Must be INVALID.NDS.CONTEXT (-1).

`ndsContext`

(IN) Provides a DSAPI context.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_Authenticate (page 198)
NMAS_Logout (page 210)

NMAS_Logout

Logs out the specified NDS eDirectory Context and closes the connection created with NMAS_LocalAuthenticate.

Syntax

```
#include <nmaspxy.h>

nint32   NMAS_Logout
(
   nint32   nmasHandle,
   nint     reserved,
   nint     ndsContext
);
```

Parameters

nmasHandle

(IN) The NMAS handle.

reserved

(IN) Must be INVALID.NDS.CONTEXT (-1).

ndsContext

(IN) Provides a DSAPI context.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_LocalAuthenticate (page 209)
NMAS_Authenticate (page 198)

NMAS_PutAttribute

Saves data in the attribute store in the directory. StoreData placed in the attribute store can be retrieved later using NMAS_GetAttribute.

Syntax

```
#include <nmaspxy.h>

 nint32                      NMAS_PutAttribute
(
     nint32                   nmasHandle,
     nint32                   aid,
     void                    *reserved,
     nint32                   attributeLenInBytes,
     void                    *attributeValue
);
```

Parameters

`nmasHandle`

(IN) The NMAS handle.

`aid`

(IN) The type of attribute to be stored.

`reserved`

Reserved. Must be null.

`attributeLenInBytes`

(IN) The length of the buffer pointed to by *attributeValue*.

`attributeValue`

(IN) The data to be stored.

Remarks

The *aid* parameter specifies the type of attribute to be stored. The *attributeValue* parameter should point to a buffer containing the data.

The values the *aid* parameter are defined in *maf.h* (see the complete list, Section 4.2, "NMAS Attribute IDs," on page 47.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

See Also

NMAS_GetAttribute (page 206)

NMAS_WhatNext

Invokes the LSM of the next method in the login sequence (if any) and returns the Method ID.

Syntax

```
#include <nmaspxy.h>

nint32   NMAS_WhatNext
(
   nint32             nmasHandle,
   MethodIDsBuffer   *doMethods,
   nint32            *mh
);
```

Parameters

nmasHandle

(IN) The NMAS handle.

doMethods

(OUT) Pointer to a Method ID's buffer structure that contains the Method ID of the login method to be invoked.

mafHandle

(IN/OUT) The MAF handle.

Return Codes

Returns NMAS_E_SUCCESS (0) if successful or a non-zero NMAS Error Code if not successful.

Remarks

If this call is successful and the entire *methodId* member of *doMethods* is zero, then there are no more methods to invoke.

If the login sequence indicates that another method is to be used, this function invokes the next method's Login Server Module.

See Also

NMAS_CanDo (page 200)
NMAS_InvokeMethod (page 208)
NMAS_Authenticate (page 198)
NMAS_LocalAuthenticate (page 209)

C.4 Secure Workstation Login Method

NMAS 2.1 ships with a generic smart card login method called Universal Smart Card. This method uses a standardized cryptographic token interface (PKCS#11) to read an X.509 certificate from a smart card, then verifies the certificate against a trusted root certificate that has been imported into eDirectory.

The secure workstation login method contains a policy that specifies (using a a method identification number) which device it should monitor for removal events. The policy also specifies other information, such as a time out for user inactivity and the "lock action" that should be taken when a login device is removed or when the inactivity time out expires.

C.4.1 Device Monitor Plug-in

The NMAS 2.1 version of Universal Smart Card includes a secure workstation plug-in that can detect when smart cards are removed and subsequently log off users from the workstation.

If you develop your own secure workstation plug-in, you need to compile a DLL that exports three functions. For a full explanation about these functions, see "Implementing A Secure Workstation Plug-in" on page 214.

The DLL you compile should monitor the login device you specify from the time StartDeviceMonitorPlugin (page 166) is called until StopDeviceMonitorPlugin (page 167). However, the Secure Workstation Policy determines whether your StartDeviceRemovalPlugin function is called at all. (See Step 1 on page 214.)

To implement your own Secure Workstation plug-in, see "Implementing A Secure Workstation Plug-in" on page 214. Also see the sample code for implementing a user ID plug-in (../../../samplecode/nmas_sample/nmas_client/idplugin.c.html).

C.4.2 Secure Workstation Policies

The Secure Workstation method uses two different policies:

- "Local Workstation Policy" on page 213
- "eDirectory Policy" on page 214

Local Workstation Policy

The local workstation policy is stored in an access control list-protected key in the registry. Secure Workstation enforces this policy as long as the service is running. No Client32 login is required to put the local workstation policy into effect. Secure Workstation starts enforcing this policy as soon as it gets a "Shell Start Event" from Winlogon.

eDirectory Policy

The eDirectory policy is configured using ConsoleOne and delivered to the workstation using the Secure Workstation Post-Login method (PLM). A separate policy can be configured for each login sequence that contains the Secure Workstation PLM.

When a user logs in using the Secure Workstation PLM, the PLM delivers the eDirectory policy to the Secure Workstation Service. The service reads the local workstation policy from the registry, then creates an effective policy that contains the most secure settings for each policy. The effective policy is enforced until the user logs out of Windows or another user logs in to eDirectory using the PLM.

Implementing A Secure Workstation Plug-in

Use the following procedure to implement your own secure workstation plug-in:

1. Compile a DLL that exports the following functions:
   - RegisterDeviceMonitorPlugin (page 165)
   - StartDeviceMonitorPlugin (page 166)
   - StopDeviceMonitorPlugin (page 167)
   - LPFN_DeviceRemoved (page 160)

   TIP: You can include this DLL when you create your NMAS LCM DLL. For more information, see Section 3.5.2, "Building a Windows LCM," on page 35.

2. Register your DLL with the Secure Workstation service by creating a registry key under the following key:

   While it does not matter what you call you registry key, this key should contain the following values:

Key	Value	Type	Description
HKLM\SOFTWARE\Novell\NMAS\Method Data\Secure Workstation\Registered Methods	MethodID	DWORD	The NMAS method ID number for your method.
HKLM\SOFTWARE\Novell\NMAS\Method Data\Secure Workstation\Registered Methods	Removal DLL	String	The path and name of the DLL that implements the Secure Workstation APIs listed above.

   Because your DLL is loaded by a service running as the LocalSystem account, you should set a restrictive access control list (ACL) on your registry key. An ACL that gives full control to the system account and read-only access to the users group is acceptable.

   NOTE: While the Secure Workstation method supports Terminal Services remote clients, the device removal functionality does not. In this release, the Secure Workstation service calls your DLL only for the user on the local console. Device removal support for Terminal Services clients might be added in a future release.

D Revision History

The following table summarizes changes made to NMAS documentation:

April 16, 2012	◆ Added error codes 16055, -16056, -16057, -16058, and -16059 to the list of NMAS error codes in Appendix A, “NMAS Error Codes,” on page 173.
October 14, 2008	◆ Modified information in “Config Store/Secret Store Attributes” on page 22.
May 27, 2008	◆ Added the methodID information to the following functions: nmasldap_put_login_config, nmasldap_delete_login_config, nmasldap_get_login_config, nmasldap_put_login_secret, nmasldap_delete_login_secret. ◆ Added or revised the following APIs: nmasldap_get_password_status, nmasldap_policy_refresh, nmasldap_check_login_policy, nmasldap_set_address_policy, nmasldap_get_user_random_password, nmasldap_get_random_password. ◆ Added information stating that the LCM and LSM need to be threadsafe.
October 17, 2007	◆ Added the following NMAS functions: ◆ NMAS_Login (page 113) ◆ Reorganized some of the MAF Password Functions (page 52) concepts to improve clarity. ◆ Updated “Placement of Files” on page 30 to document the proper directory structure for LCM, LSM, and license files. ◆ Updated the directory structure in Section 3.5.3, “Building a Linux LCM,” on page 35. ◆ Changed Section 4.3.8, “MAF Data Store Functions***,” on page 52 from Early Access to Supported Status. ◆ Updated the NMAS Password Manager Java classes to document the constructor that take a JLdapTransport object. See Chapter 9, “NMAS Javadoc References,” on page 171. ◆ Updated Appendix A, “NMAS Error Codes,” on page 173.
February 28, 2007	◆ Added Section 2.2.1, “NMAS Login Method Security Considerations,” on page 16. ◆ Added the function, MAF_GetPasswordEx (page 63).
October 11, 2006	◆ Corrected the RestartPluginOnFailure registry setting. ◆ Fixed broken links.

June 21, 2006

- Updated `nmasext.h` to enable C++ functionality.
- Documented the "Configuration Store Functions" on page 107:
  - nmasldap_delete_login_config (page 119)
  - nmasldap_get_login_config (page 123)
  - nmasldap_put_login_config (page 125)
- Documented the "SecretStore Management Functions" on page 107:
  - nmasldap_delete_login_secret (page 121)
  - nmasldap_put_login_secret (page 127)
- Revised or added the following NMAS 3.1 Leading Edge functions:
  - MAFDS_ATTRIBUTE (page 87)
  - MAFDS_FreeContainerEntries (page 89)
  - MAFDS_FreeModValues (page 91)
  - MAFDS_FreeValues (page 92)
  - MAFDS_FreeValueData (page 93)
  - MAFDS_GetParentContainer (page 94)
  - MAFDS_GetPartitionRootContainer (page 95)
  - MAFDS_GetValueData (page 96)
  - MAFDS_InsertModValue (page 97)
  - MAFDS_ListContainerEntries (page 99)
  - MAFDS_ModifyEntry (page 100)
  - MAFDS_ReadAttributeValues (page 102)
  - MAFDS_ReadInheritedAttributeValues (page 104)
- Linked NMAS Javadoc classes directly from the on line NMAS developer documentation.
- Corrected some Error Code definitions.

March 1, 2006	◆ Updated Novell documentation templates and fixed broken links.
	◆ Corrected the objectDN parameter documentation in all of the NMAS LDAP functions to reflect that it is an LDAP DN.
	◆ Added MAF_MemMalloc (page 69), MAF_MemMalloc (page 69), and MAF_MemFree (page 68) to replace MAF_Malloc (obsolete 3/1/2006) (page 67) and MAF_Free (obsolete 3/1/2006) (page 58).
	◆ Added new MAF_LogEvent (page 65) function to add logging functionality in NMAS 3.0 or later.
	◆ Updated "NMAS Error Codes" on page 173.
	◆ Added theSection 4.3.8, "MAF Data Store Functions***," on page 52:
	    ◆ MAFDS_CreateContext (page 88)
	    ◆ MAFDS_FreeContext (page 90)
	    ◆ MAFDS_GetValueData (page 96)
	    ◆ MAFDS_InsertModValue (page 97)
	    ◆ MAFDS_ListContainerEntries (page 99)
	    ◆ MAFDS_ModifyEntry (page 100)
	    ◆ MAFDS_ReadAttributeValues (page 102)
	    ◆ MAFDS_VALUE_DATA (page 106)

October 5, 2005	◆ Transitioned to revised Novell documentation standards. ◆ Revised Chapter 5, "Method Management Functions," on page 107 and Chapter 7, "Password Management Functions," on page 135 to include new universal password management Java methods.
June 1, 2005	◆ Totally revised and reorganized the NMAS NDK to help facilitate implementation of the new Java interface. ◆ Added NMAS NDK Java documentation (../api/index.html) for NMAS login interfaces. ◆ Expanded Error Code definitions to accommodate new NMAS functionality. ◆ Added new NMAS Sample Code (http://developer.novell.com/ndk/doc/samplecode/nmas_sample/index.htm) and enhanced existing samples to help demonstrate NMAS method development. ◆ Deleted the NMAS_ActivateLockLogout function, which was deprecated in 2002.
March 2, 2005	◆ Created a new appendix to document the Deprecated NMAS Functions (page 187). ◆ Added the Disconnected Mode concept documentation, which was never supported in the NMAS SDK. ◆ Made technical corrections and fixed broken links. ◆ Deleted unsupported references to SASL Authentication as a method for adding authentication support to connection-based protocols.

October 6, 2004	◆ Added documentation to support new LDAP password management functions: NMAS Password Management Java Classes (page 135) Password Management Requirements (page 136) Simple Password Management (page 136) Universal Password Management (page 137) NMAS LDAP C Password Management Functions (page 138) ◆ Added the following new Sample Code (http://developer.novell.com/ndk/doc/samplecode/nmas_sample/index.htm) files: Universal Password Simple Password Login Configuration Login Secret ◆ Made technical corrections and fixed broken links.
July 23, 2004	◆ Clarified when these routines can be called: NMAS_Authenticate (page 198) NMAS_AuthenticateConnection (page 199) NMAS_LocalAuthenticate (page 209)
June 9, 2004	◆ Updated libraries and software to incorporate bug fixes to version 2.33. ◆ Made minor modifications and fixed broken links.
February 18, 2004	◆ Fixed a number of bugs in software and documentation. ◆ Added the following flags to NMAS Attribute IDs (page 47): NMAS_AID_OPTIONS, NMAS_AID_FEATURES, NMAS_AID_PWD_WARNING, and NMAS_AID_VERIFY_ONLY.
June 2003	◆ Updated NMAS Error Codes (page 173). ◆ Added the following Multiple Authentication Framework Functions: MAF_AllowPasswordSet (page 55) MAF_GetPassword (page 61) MAF_Free (obsolete 3/1/2006) (page 58) MAF_SetPassword (page 75) MAF_Trace (page 76) MAF_TraceEnabled (page 77) MAF_TraceOnError (page 78) ◆ Updated NMAS Attribute IDs (page 47). ◆ Edited entire document for style.
March 2003	◆ Added new references to “SASL Authentication Support” as a method for adding authentication support to connection-based protocols. ◆ Added new section, Implementing A Secure Workstation Plug-in (page 214). ◆ Revised Client Application Login Functions (page 187) to explain how NMAS client application login functions are used by Windows applications. ◆ Added new NMAS_LOGIN_IDENTITY (page 159) function to replace NMAS_GetUserName (obsolete 3/03) (page 168). Consequently, made major revisions to the entire Identification Method Function chapter.
September 2002	Revised Client Application Login Functions section to explain how NMAS client application login functions are used by Windows applications.
June 2002	Mid-release update of documentation of proxy functions to correlate with changes made to the NMAS software and sample code, which was previously submitted for the May 2002 NDK release.

May 2002	Updated NMAS Error Codes (page 173) section, as well as signing code and login method sample code examples.
February 2002	Added NMAS Error Codes (page 173) section and added new login Identification Method chapter. Enhanced Sample Code and corrected several documentation errors.
October 2001	Clarified a number of concepts and added the following new Proxy Functions: ◆ NMAS_FindLoginSequences (page 205). ◆ NMAS_FindMethods (page 204). ◆ NMAS_GetAvailableMethods (page 207). ◆ NMAS_AuthenticateConnection (page 199).
June 2001	Added Chapter 5, “Method Management Functions,” on page 107. Revised Section 4.2, “NMAS Attribute IDs,” on page 47. Edited and enhanced Sample Code.
February 2001	Added the Disconnected Mode concept. Added NMAS_AID_DISCONNECTED_FLAG to the list of constants that can be passed as a parameter in the MAF_GetAttribute (page 59) function. Revised the Return Codes sections of the following functions to explain that they will return NMAS_E_NOT_SUPPORTED while in disconnected mode: MAF_Read (page 73), MAF_XRead (page 82), MAF_Write (page 79), MAF_XWrite (page 84), MAF_WriteRead (page 80), MAF_XWriteRead (page 85). Added **nmasapi.h** to the list of files needed in the Building a Windows LCM (page 35) and Tasks for Writing a Login Method (page 23) tasks.
September 2000	Added requirements for writing Login Server Methods and Login Client Methods for Windows NT server. See Building a Windows LCM (page 35) and Steps for Signing an LSM (page 26).
May 2000	Added Glossary chapter and linked definitions within the documentation. Updated Multiple Authentication Framework Functions (page 45), Method Management Functions (page 107), and Proxy Functions (page 196) reference information.

Glossary

Authentication Store. Various eDirectory™ class attributes where secret authentication information is stored. These stores can be class attributes of the Login Method object or User objects.

biometric. Biological measurements of the eyes, voice, fingerprints, and face used for logging in and authenticating to the network.

For more information, see Understanding Authentication Methods (http://www.novell.com/documentation/lg/usfwnt3/docui/index.html#../fwcn_enu/data/hcjh3vxj.html).

clear text password. Passwords that have not been encrypted. In contrast, *ciphertext* is data that has been encrypted.

configuration data. Data that can be stored and retrieved through the Method Management Functions. In addition, the MAF_GetAttribute function of the Multiple Authentication Framework Functions can read this data to get the value of an attribute associated with the current login session. The MAF_PutAttribute can store/delete this data.

Because the application can store and retrieve any values, configuration data is encrypted and secured within eDirectory. The configuration data storage area is defined as a multiple value attribute that can be associated with any user object, Login Method object, or Login Device Object within eDirectory.

Directory Services Trace. DSTrace. A troubleshooting aid to help debug problems with eDirectory and projects that utilize Novell® Directory Services (NDS®). For more information to understand or implement DSTrace, see the following references: How to Use DSTrace (http://support.novell.com/cgi-bin/search/searchtid.cgi?/2908733.htm), More On Using the DSTRACE Command (http://developer.novell.com/research/sections/netmanage/dirprimer/2001/septembe/spv.htm), or Looking Into the DSTrace Options (http://developer.novell.com/research/sections/netmanage/dirprimer/2001/august/spv.htm).

eDirectory. A Novell distributed, replicated naming service that maintains information about and provides access to every resource on the network. eDirectory tightly integrates Novell Security Services for e-commerce (PKI, cryptography, and authentication services), allowing developers to build applications that can be accessed and managed across the entire network through explicit policies.

GA. Graded Authentication. A module that controls access to information based on how a user has authenticated to a system. GA associates varying *clearances* to connections on the basis of network policy, such as the authentication protocols and methods used, the properties of the workstation, or the requested capabilities.

For more information, see Understanding Authentication Methods (http://www.novell.com/documentation/lg/usfwnt3/docui/index.html#../fwcn_enu/data/hcjh3vxj.html)

Grade. The login properties of a login method. Currently, biometric, password, token, and logged (or any combination) of grades are supported. See Login Method Overview.

LCM. Login Client Module. A client-side component of a login method. The LCM is essentially a program running on the workstation that interacts with the LSM. The LCM and LSM transmit login credentials using the Multiple Authentication Framework Functions. See Building an NMAS LCM.

LDAP. Lightweight Directory Access Protocol. An X.500-related Open Systems Interconnection (OSI) protocol that clients can use to read and write directory information. LDAP is used to publish directory information.The directory features available to LDAP clients are dependent upon the features built into the LDAP server and the LDAP client; some clients have the ability to read and write data, others can only read directory data.

Login Device Object. Object contained within a Login Method object. An LD is neither managed nor created by NMAS. If used, LD's are created and managed by the MMG (Method Management Graphical Interface).

login method. Plug-in modules for the NMAS™ framework. Login methods are used to identify a user to eDirectory for authentication. A login method includes a Method Management Graphical Interface (MMG), a Login Client Module (LCM), and a Login Server Module (LSM).

For more information, see Understanding Authentication Methods (http://www.novell.com/documentation/lg/usfwnt3/docui/index.html#../fwcn_enu/data/hcjh3vxj.html)

Login Method container. An eDirectory object that is contained in the Security container at the tree root. The login method container class contains a Login Method object for each login method.

The Post Login Method container is contained in the Security container. Post Login Method objects are contained in the Post Login Method container. The Login Policy object is contained in the Security container. See "Authorized Login Method Container (LMC)" on page 20.

Login Method object. Object contained in the Login Method container (see Authentication Store). Every login method requires a unique Login Method object.

Login Secret Data. Secret user authentication data that can only be written through the MMG ("Method Management Functions" on page 107) or with the MAF_PutAttribute function. However, secret data can only be read by the LSM through the MAF_GetAttribute function.

Secret Data is secured with an authentication strength encryption. The secret data storage area is defined as a multiple value attribute that is associated with any User object, Login Method object, or Login Device Object within NDS.

login sequence. An ordered list of login and post login methods. Login sequences are stored in the Login Policy object. There are two types of login sequences: AND and OR. For an AND sequence to be successful, all login methods must validate the user. For an OR sequence to be successful, only one of the login methods must validate the user. In either, all post login methods must complete successfully.

LSM. Login Server Module. The server-side component of a login method. The LSM is invoked by the NMAS Server Manager. The LSM works with the LCM to transmit login credentials using the Multiple Authentication Framework Functions in a client-server model. For more information, see Section 3.1, "Building an NMAS Login Method," on page 23.

MAF. Multiple Authentication Framework Login Protocol. A protocol used for communication between the LCM and the LSM, which establishes login credentials in NMAS. For more information, see the Multiple Authentication Framework Functions and the Login Method Overview.

message digest. A data string distilled from the contents of a text message, created using a one-way hash function. Encrypting a message digest with a private key creates a digital signature, which is an electronic means of authentication.

MMG. Method Management Graphical Interface. Java snap-in application designed to operate within the Novell ConsoleOne® management framework. An MMG snap-in application for a login method lets the administrator set up login parameters and manage the credential data that is stored in the Authentication Store.

NMAS provides the Method Management Functions that allow the MMG to access the Authentication Store. See "Method Management Functions" on page 107.

NICI. Novell International Cryptography Infrastructure. A policy-based architecture, providing enforced key usage and key archiving based on dynamically loadable policies. The application of NICI is independent of, and yet controlled by, the internal governing policies. This means that restrictions on the use of cryptography are transparent to the application developer.

NICI is a modular architecture, supporting replaceable cryptographic engines to provide for special cryptographic needs, both present and future. Using NICI, developers receive the benefits of flexible and current technologies without modifying their software products.

NMAS. Novell Modular Authentication Services. A flexible and expandable login framework that provides developers the ability to integrate multiple authentication services using NICI into eDirectory systems.

NMAS works with the Graded Authentication (GA) capability first shipped with NetWare® 5.*x*, providing a common point of administration for all login methods and policies through NDS®.These login methods are used to identify a user to eDirectory. A complete login method includes a Method Management Graphical Interface (MMG), a Login Client Module (LCM), and a Login Server Module (LSM).

NMAS Client Manager. An interface for other applications' transports to perform login sessions (see "Method Management Functions" on page 107). NetWare Client32™ invokes the NMAS Client Manager after the login screen is completed.

The NMAS Client manager is responsible for creating an NMAS session for the current session, initializing the NICI Client, establishing a session key, placing the login data in the Security Context Manager as attributes that can be read by the LCM, invoking the login method, and returning the status of Login/Authentication.

The NMAS Client manager also is responsible for obtaining the credentials used by the Authentication Manager. The Login Manager implements the Multiple Authentication Framework (MAF) login protocol: (1) determining the available Login Method object supported by the client, (2) initiating the MAF protocol, (3) receiving the "DO" MAF commands, and (4) invoking the appropriate LCM module(s).

Upon receipt of the "SUCCESS" MAF command, the Login Manager retrieves the credential materials from the server and places them into the Authentication Store for use by the Authentication Manager.

See NMAS Client Manager.

NMASMon. A NetWare NLM™ that provides a way to get trace information from NMAS, similar to "Directory Services Trace" on page 221 for eDirectory. For more information, see Using NMASMON (http://www.novell.com/documentation/lg/nmas21/index.html?page=/documentation/lg/nmas21/admin/data/ahl2qgo.html).

NMAS Server Manager. A module that registers with the NCP™ provider to receive the NMAS NCPs (number 94). The NMAS Server Manager is invoked when the NMAS module is loaded on the server. It also provides an interface for other transports (for example, to transport NMAS messages between the client and server). (See "Method Management Functions" on page 107).

Where application specific protocols are used for login and authentication, the NMAS Server Manager provides an API set so that Proxy Services can be implemented (see Section C.3, "Proxy Functions," on page 196).

The NMAS Server Manager is responsible for creating an NMAS session for the current session, receiving MAF requests for login and authentication, invoking the login method, and returning the status of the login process.

See NMAS Server Manager.

Novell Certificate Server. An enterprise PKI solution that offers the ability to freely mint an unlimited number of digital certificates for end users (for example, to enable secure e-mail or X.509 certificate-based authentication) and for other servers (for example, to enable SSL security). See Novell Certificate Server (http://developer.novell.com/ndk/ncslib.htm).

OCX. A short name for OLE custom controls. Such control modules end with a.ocx extension.

PKI. Public Key Infrastructure. The PKI framework used to securely exchange information, using certification authorities (CAs) and digital signatures to verify and authenticate the validity of persons engaged in Internet, Intranet, and Extranet transactions. A reliable PKI system is necessary before implementing a secure e-commerce strategy. Use the Novell Certificate Server Libraries for C (http://developer.novell.com/ndk/ncslib.htm) to implement your own internal PKI system.

Post Login method. Methods that are invoked after login has been completed. Examples are screen saver and change password. PLM has a Login Client Module (LCM), an Login Server Module (LSM), and usually a Method Management Graphical Interface (MMG).

proxy services. NMAS Server provides Login and Authentication functionality for various services. Some of the proposed proxy methods can include SSL, HTTP, RADIUS, and LDAP. The proxy support is provided through Section C.2, "NMAS Transport API," on page 190 that are exposed through the NMAS Server Manager. These Proxy Functions allow the service to emulate the NMAS Client so that it can work in conjunction with NMAS Server components.

SASL. Simple Authentication Security Layer. A method for adding authentication support to connection-based protocols. To use this specification, a protocol includes a command for identifying and authenticating a user to a server and for optionally negotiating a security layer for subsequent protocol interactions. It is used with popular Internet protocols such as POP3, IMAP, SMTP, and LDAP.

The LDAP v3 protocol uses the SASL to support pluggable authentication. This means that the LDAP client and server can be configured to negotiate and use possibly nonstandard and/or customized mechanisms for authentication, depending on the level of protection desired by the client and the server. The LDAP v2 protocol does not support the SASL.

RADIUS. Remote Authentication Dial-In User Service. A Novell authentication service that enables remote users to securely dial in to NetWare networks (version 5.1 or later) and access network information and resources. When installed on a server, RADIUS can be configured as an integral part of NMAS.

smart card. An electronic card containing a built-in microprocessor and memory used for authentication identification. To use a smart card with NMAS, you need a smart card reader and an installed NMAS method built for that reader.

SSL. Secure Socket Layer. The set of rules governing the exchange of information between two devices using a public key encryption system. SSL establishes and maintains secure communication between SSL-enabled servers and clients across the Internet.

X.509 Certificates. The common standard used for creating digital certificates. However, companies have implemented the X.509 standard in different ways, rendering some generated X.509 certificates unreadable across software products from different companies. A widely-used specification for digital certificates that has been a recommendation of the ITU since 1988. See X.509 Property Page (http://www.novell.com/documentation/lg/nw5/docui/index.html#../ussecur/crndsenu/data/h0000070.html).