Novell Developer Kit

1.0

June 15, 2005

forge.novell.com

COMMON AUTHENTICATION SERVICE ADAPTER (CASA)

Legal Notices

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.

www.novell.com

NDK: Common Authentication Service Adapter
June 15, 2005

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

Novell Trademarks

Novell is a registered trademark of Novell, Inc. in the United States and other countries.
SUSE is a registered trademark of SUSE AG, a Novell business.

Third-Party Materials

All third-party trademarks are the property of their respective owners.

Contents

About This Guide

Common Authentication Service Adapter (CASA) provides a common infrastructure for client authentication across the Linux* and Microsoft* Windows* desktops. Novell® products such as GroupWise®, GroupWise Messenger, iPrint, Novell iFolder®, and the Novell clients for Windows and Linux are integrated with the miCASA interface and can take advantage of the credential store that provides the cornerstone for CASA.

This guide contains the following sections:

User Comments

We want to hear your comments and suggestions about this manual. To contact us, send an e-mail message to ndk@novell.com (mailto:ndk@novell.com).

1 Overview

Applications that require credentials also require some type of credential management logic. The miCASA framework provides applications a place to securely store their credentials and the ability to share those credential with other applications. This reduces the number of credentials being managed, and provides a single sign-on experience to the end user.

This document describes the API set provided by the miCASA framework, as well as the logic used internally for applications to share common credentials.

A credential stored in the miCASA framework is given a unique name by the application developer, known as the appSecretID. Currently, a credential consists of a username and a password. The username can be any of the following forms:

- Common Name (CN), for example John Smith.
- Distinguished Name (DN_NDAP), for example, admin.*novell*.
- Fully Distinguished Name (FDN_NDAP), for example like cn=admin.o=*novell*.
- Fully Distinguished LDAP Name (DN_LDAP), for example cn=admin,o=*novell*.

The miCASA framework is capable of storing all of these forms under the same credential name or appSecretID. This is done as key-value pairs in a data buffer as shown here:

```
CN=admin<LF>Password=mySecret<LF>DN_NDAP=admin.novell<LF>...
```

This data buffer is known as a Credential Set, or SS_CredSet. The data in the buffer is managed by the miCASA API. The application developer does not need to parse this data directly, but simply passes the username and password in the defined structure defined in the API.

The appSecretID should be unique for each application using the miCASA API. For example, we suggest the following naming convention:

```
Company.ApplicationName (for example, Novell.Groupwise or Novell.IFolder)
```

If the application needs to store more than one credential, it can append additional strings to the end of the appSecretID.

Sharing Credentials

Credentials used by an application authenticate against some type of realm. This realm might be an NDS Tree, an Active Domain, a managed database, or even a combination of all of these. The Authentication Realm is defined by the network administrator. It is common for multiple applications to authenticate to the same realm.

miCASA APIs provide for the sharing of such credentials. If the application can discover or can be configured to use the name of an Authentication Realm, the miCASA provides the application a mechanism to share this credential. The miCASA API set described later provides a sharedSecretID parameter for which application developers can leverage each others credential

sets. This parameter is not required but assists the API in accessing the proper credential. Novell® iPrint™ is an example of such an application that discovers the Tree name or authentication realm of a chosen network printer.

NOTE: the miCASA framework is written with the ability for the user or network administrator to override the sharedSecretID used by a given application. However, management of the feature is not yet functional.

2 CASA on Linux

This is your guide to using the Common Authentication Service Adapter (CASA) developer kit on Linux.

This section covers the following topics:

For information on using CASA with Microsoft* Windows*, see Chapter 3, "CASA on Windows," on page 15.

Linux Components

The main components of CASA on Linux are:

1. **CASA Identity Development Kit (IDK):** The IDK provides a set of APIs that application and service developers can use to write user/application credentials to the credential store. The IDK APIs internally store the credentials passed onto them by the applications in miCASAd. There are C, C++, C# and Java bindings available for the CASA IDK.
2. **miCASAd:** An active component that starts during boot time. On Linux, miCASAd is available in the run-levels 1, 2, 3 and 5. It runs with root privileges and is active as long as the system is up. It stores and provides credentials or secrets based on the Linux user identifier (uid) of the process that makes the IDK API calls. The credentials, which are stored by applications in miCASAd, are maintained only in memory for the first release. Session-based secrets implies secrets that are stored in an in-memory cache, are available only as long as the user is in session on the desktop, and are destroyed when miCASA daemon is restarted or the user logs off.

   NOTE: Any PAM module that uses the IDK APIs must set its effective user id temporarily to that of the user logging in (the user returned by calling pam_get_user()), if the credentials need to be stored against that user. There might be cases where the user obtained through pam_get_user() might not be the one against whom the PAM module actually intends to store credentials.
3. **Login Credential Capture Module:** On Linux, the login credential capture module is implemented as a PAM module. This PAM module captures the user's desktop login credentials and stores them in miCASAd using the IDK APIs. This PAM module is placed as the last module in the auth and session stacks of xdm, gdm, kdm, login and sshd PAM configuration files. In the auth stack, the functionality of this module is to store the credentials in miCASAd and in the session stack, then closes the user's session with miCASAd.

CASA Linux Packages

CASA consists of two Linux packages:

- **CASA-1.0.0.i586.rpm:** Installs miCASAd, the startup scripts, the Login Credential Capture PAM module, and the relevant libraries required by any application using the CASA APIs to function.
- **CASA-devel-1.0.0.i586.rpm:** Installs the relevant header files that developers need to write applications to the CASA APIs. This is dependent on CASA-1.0.0.i586.rpm

Except /opt/novell/CASA/include directory, which is the only directory installed by CASA-devel-1.0.0.i586.rpm, all other directories are installed by CASA-1.0.0.i586.rpm.

Linux Directories and Files

CASA Linux files are located in the following directories:

/opt/novell/CASA/doc

The /opt/novell/CASA/doc directory contains the following files:

File	Description
CASA_Reference_Guide.pdf	This document.
README.txt	The readme file, which contains information about any last-minute updates.

/opt/novell/CASA/lib or /opt/novell/CASA/lib64

This directory contains the following files for 32-bit machines (/opt/novell/CASA/lib) or 64-bit machines (/opt/novell/CASA/lib64):

File	Description
libmicasa.so.*	The miCASA C/C++ developer kit library.
miCASA.jar	The miCASA Java* developer kit jar file.
libjmicasa.so.*	The miCASA Java* developer kit library.
Novell.CASA.miCASAWrapper.dll	The miCASA C# developer kit library, which is based on Mono®.

/lib/security or /lib64/security

This directory contains the following file for 32-bit machines (/lib/security) or 64-bit machines (/lib64/security):

File	Description
pam_micasa.so	The miCASA login credential capture module that is inserted in the auth and session stacks of the PAM configuration files of xdm, gdm, kdm, login and sshd.

/opt/novell/CASA/bin

The /opt/novell/CASA/bin directory contains the following files:

File	Description
micasad.exe	The miCASA daemon that starts up at run levels 1, 2, 3, and 5 (which is based on Mono).
micasad.sh	A script that starts micasad.exe, which is located in the /opt/novell/CASA/bin directory.

/etc/init.d

The /etc/init.d directory contains the following file:

File	Description
micasad	The micasad startup script. This script is started in run levels 1, 2, 3, and 5. To achieve this, there are links to this script from the appropriate run-level directories (/etc/rc1.d, /etc/rc2.d, /etc/rc3.d and /etc/rc5.d). This script calls the /opt/novell/CASA/bin/micasad.sh script to start the daemon.

/opt/novell/CASA/include

The /opt/novell/CASA/include directory contains the following files:

File	Description
micasa.h	The low-level header file that lists the C/C++ functions.
micasa_mgmd.h	The main header file for C/C++ developers.

Using CASA with Linux

This section covers the following topics:

Linux Installation

CASA is preinstalled on the Novell Linux Desktop SP2 operating system.

On other distributions, use the following to install CASA:

```
rpm -Uvh CASA-1.0.0.i586.rpm
rpm -Uvh CASA-devel-1.0.0.i586.rpm
```

Linux Start Up

Use the following command to start, stop, and restart the CASA service:

```
/etc/init.d/micasad [start|stop|restart]
```

Linux Uninstallation

Use the following commands to uninstall the CASA packages:

```
rpm -e CASA-devel
rpm -e CASA
```

3 CASA on Windows

This is your guide to using the Common Authentication Service Adapter (CASA) developer kit on Microsoft Windows.

This section covers the following topics:

- "Windows Components" on page 15
- "Using CASA with Windows" on page 17

For information on using CASA with Linux, see Chapter 2, "CASA on Linux," on page 11.

Windows Components

CASA consists of one Windows package, CASA-1.0.0.msi, which is the installation module that contains the following two components that match their Linux counterparts (see "Linux Components" on page 11):

- CASA-devel-1.0.0.msm
- CASA-1.0.0.msm

Windows Directories and Files

CASA Windows files are located in the following directories:

- "\Program Files\Novell\CASA\bin" on page 15
- "\Program Files\Novell\CASA\include" on page 16
- "\Program Files\Novell\CASA\lib" on page 16
- "\Program Files\Novell\CASA\doc" on page 16
- "\windows\system32(64)" on page 16

\Program Files\Novell\CASA\bin

The \Program Files\Novell\CASA\bin directory contains the following files:

File	Description
DisplayCache.exe	A tool for dumping the cache.
Lcredmgr.dll	The login capture, login extension, and logout for Novell® Client32™.
micasad.exe	The miCASA service for Windows.

File	Description
sshtst.exe	The tool to test the miCASA service.

\Program Files\Novell\CASA\include

The \Program Files\Novell\CASA\include directory contains the following files:

File	Description
micasa.h	The low-level header file that lists the C/C++ functions.
micasa_mgmd.h	The main header file for C/C++ developers.

\Program Files\Novell\CASA\lib

The \Program Files\Novell\CASA\lib directory contains the following files:

File	Description
micasa.lib	The miCASA C/C++ developer kit front-end dynamic library for linking.
miCASA.jar	The miCASA Java developer kit jar file.
Novell.Security.ClientPasswordManager.NetCredential.dll	The .Net wrapper for Novell iFolder® access.
Novell.CASA.miCASAWrapper.dll	The miCASA C# developer kit library, which is based on .Net.
Novell.Security.Utilities.dll	The miCASA utility for debug logging.

\Program Files\Novell\CASA\doc

The \Program Files\Novell\CASA\doc directory contains the following files:

File	Description
CASA_Reference_Guide.pdf	This document.
README.txt	The readme file, which contains information about any last-minute updates.

\windows\system32(64)

The \windows\system32(64) directory contains the following files:

File	Description
micasa.dll	The miCASA C/C++ developer kit dynamic library.
micasacache.dll	The miCASA library that allows the developer kit to talk to the miCASA service.

File	Description
jmicasa.dll	The miCASA JNI library for the Java interface.

Using CASA with Windows

This section covers the following topics:

Windows Installation

To install CASA on the Windows operating system, double-click the CASA.msi file.

Windows Startup

After you have installed CASA, you can use the Start > Control Panel > Administrative Tools > Services window to start the Novell Identity Store service.

You can also reboot your machine to start the service.

Windows Uninstallation

To uninstall CASA, choose CASA from the Start > Control Panel > Add/Remove Programs window and follow the instructions.

4 Functions

The following functions allow an application that requires credentials to get, set, and clear a credential:

- “miCASAGetCredential” on page 20
- “miCASARemoveCredential” on page 21
- “miCASASetCredential” on page 22

All strings must be NULL terminated, and their length must include the NULL byte.

For a list of possible error codes, see the micasa_mgmd.h header file.

miCASAGetCredential

Allows an application to get a credential.

Syntax

```
int miCASAGetCredential
(
   uint32_t           ssFlags,
   SSCS_SECRET_ID_T  *appSecretID,
   SSCS_SECRET_ID_T  *sharedSecretID,
   int32_t           *credentialType,
   void              *credential,
   SSCS_EXT_T        *ext
);
```

Parameters

ssFlags

(IN) Set to 0 for this release.

appSecretID

(IN) Points to a structure of a unique string that represents the name of the service that is requesting the credentials, such as Novell.GroupWise, Novell.iFolder.

sharedSecretID

(IN) Optional. Points to a structure of the shared name of the back end authentication realm that relates a group of services. This ID allows multiple applications to find and store a shared credential, such as Novell_Collaboration. You can set this parameter to NULL.

credentialType

(IN/OUT) Points to the type of credential that is being used.

credential

(OUT) Points to the credential structure.

ext

Reserved for future use.

Remarks

miCASAGetCredential returns the requested credential by using the following steps:

1. Returns the credential for the sharedSecretID, if one is requested, and found.
2. Returns the credential for the appSecretID, if the sharedSecretID is not found or not requested.
3. Returns the default credential if 1 and 2 above fail.
4. Returns an error if 1,2 or 3 above fail.

miCASARemoveCredential

Allows an application to remove a credential.

Syntax

```
int miCASARemoveCredential
(
   uint32_t          ssFlags,
   SSCS_SECRET_ID_T *appSecretID,
   SSCS_SECRET_ID_T *sharedSecretID,
   SSCS_EXT_T       *ext
);
```

Parameters

ssFlags

(IN) Set to 0 for this release.

appSecretID

(IN) Points to a structure of a unique string that represents the name of the credential that should be removed, such as Novell.GroupWise, Novell.iFolder.

sharedSecretID

(IN) Ignored for this release.

ext

Reserved for future use.

miCASASetCredential

Allows an application to set a credential.

Syntax

```
int miCASASetCredential
(
   uint32_t           ssFlags,
   SSCS_SECRET_ID_T  *appSecretID,
   SSCS_SECRET_ID_T  *sharedSecretID,
   int32_t           *credentialType,
   void              *credential,
   SSCS_EXT_T        *ext
);
```

Parameters

ssFlags

(IN) Specifies to persist the credentials across reboots of the application. Set to 0.

appSecretID

(IN) Points to a structure of a unique string that represents the name of the service that is requesting the credentials, such as Novell.GroupWise, Novell.iFolder.

sharedSecretID

(IN) Optional. Points to a structure of the shared name of the back end authentication realm that relates a group of services. This ID allows multiple applications to find and store a shared credential, such as Novell_Collaboration. You can set this parameter to NULL.

credentialType

(IN) Points to the type of credential that is being used.

credential

(IN) Points to the credential structure.

ext

Reserved for future use.

Remarks

NSSCSSetCredential sets the requested credential by using the following steps:

1. Sets the credential for the sharedSecretID if one is supplied.
2. Sets the credential for the appSecretID if the sharedSecretID is not supplied or if setting the sharedSecretID fails.

5 Structures

CASA uses the following structures:

SSCS_BASIC_CREDENTIAL

Contains credential information.

Syntax

```
typedef struct sscs_basic_credential
{
   uint32_t    unFlags;
   uint32_t    unLen;
   SS_UTF8_T   username;
   uint32_t    pwordLen;
   SS_UTF8_T   password;
} SSCS_BASIC_CREDENTIAL;
```

Fields

unFlags

Specifies the supported flags (see the header file).

unLen

Specifies the length of the structure.

username

Specifies the user name, with a maximum length of NSSCS_MAX_USERID_LEN.

pwordLen

Specifies the length of the password.

password

Specifies the password, with a maximum length of NSSCS_MAX_PWORD_LEN.

SSCS_SECRET_ID_T

Provides the credential information.

Syntax

```
typedef struct sscs_secret_id
{
   uint32_t    len;
   SS_UTF8_T   id;
} SSCS_SECRET_ID_T;
```

Fields

len

Specifies the length of the secretID.

id

UTF-8 string representing the secret or the credential.

Revision History

This section outlines all the changes that have been made to the Common Authentication Service Adapter (CASA) documentation (in reverse chronological order).

June 15, 2005	Revised documentation to coincide with software updates.
June 3, 2005	Posted as beta documentation.