Novell Developer Kit

1.5

November 18, 2005

www.novell.com

COMMON AUTHENTICATION SERVICE ADAPTER (CASA)

Novell®

Legal Notices

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

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

Novell Trademarks

Third-Party Materials

Contents

About This Guide

The Common Authentication Service Adapter (CASA) SDK provides a common authentication and security package 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:

Audience

This guide is intended for advanced application developers who want to enable single sign-on to an enterprise network. In order to deploy this API on your applications, you should be familiar with Linux and Windows development platforms, as well as an understanding authentication and security development concepts.

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 *CASA SDK*, visit the Novell Common Authentication Service Adapter Web site (http://developer.novell.com/ndk/casa.htm).

Additional Documentation

For documentation on other authentication and SecretStore issues, see the Novell SecretStore product documentation (http://www.novell.com/documentation/secretstore33/index.html) and the Novell SecretStore Developer Kit for C (http://developer.novell.com/ndk/ssocomp.htm) Web sites. The CASA SDK replaces the SecretStore Developer Kit.

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.

Getting Started

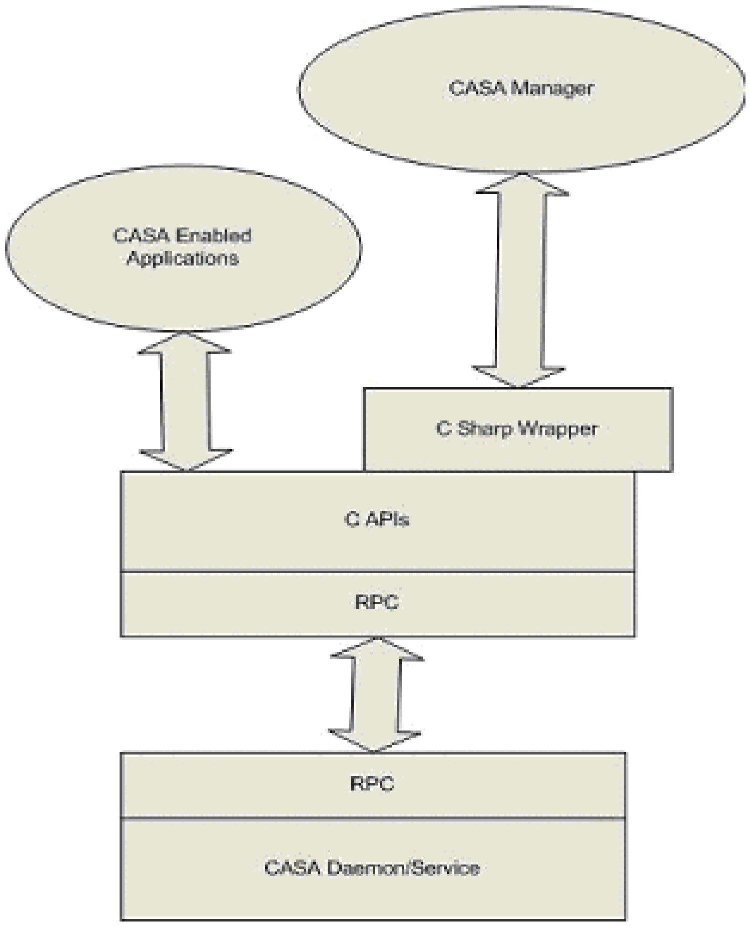
1

The Novell Common Authentication Services Adaptor (CASA) is a common authentication and security package that provides a set of libraries for application and service developers to enable single sign-on to an enterprise network. CASA 1.5 provides a local, session-based credential store, called miCASA, that is populated with desktop and network login credentials on the following workstations:

- Novell Linux Desktop (NLD SP2)
- Windows XP Home/Professional
- Windows 2000 Professional

As shown in the following architectural diagram, within the CASA framework, the miCASA credential store is the component you incorporate and enable on your applications.

Figure 1-1 *CASA Architectural Structure*



The miCASA credential service is implemented in C# with API bindings in C, C#, and JAVA. CASA also provides a Network Credential class to enable single sign-on in .NET framework applications

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 credentials with other applications. This reduces the number of credentials that are being managed and provides a single sign-on experience to the end user.

This document describes the functions provided by the miCASA framework, as well as the logic used internally that allows applications to share common credentials:

- Section 1.1, "Credentials," on page 12
- Section 1.2, "Sharing Credentials," on page 12

1.1 Credentials

A credential stored in the miCASA framework is given a unique name known as the SecretID. 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, 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 SecretID. This type of secret is known as a Credential Set, or SS_CredSet.

The SecretID 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 your application needs to store more than one credential, you can append additional strings to the end of the SecretID.

1.2 Sharing Credentials

Credentials that are used by an application authenticate against some type of realm. This realm might be an eDirectory™ Tree, an Active Directory* domain, a managed database, or even a combination of all of these. The network administrator defines the Authentication Realm and multiple applications commonly authenticate to the same realm. The miCASA functions enable applications to share such credentials.

Discovering the Realm

In order for credential sharing to take place, your application either must be able to discover the Authentication Realm or be configured to use the name of the Authentication Realm.

The miCASA API functions described in this document provide a sharedSecretID parameter that you can use to leverage credential requirements of applications used within the realm. Although not required, this parameter assists the API in accessing the proper credential. Novell® iPrint is an example of an application that discovers the Tree name or authentication realm of a chosen network printer.

NOTE: The miCASA framework is designed so that the user or network administrator can override the sharedSecretID that is used by a given application. However, this feature is not yet functional.

CASA on Linux 2

- Section 2.1, "Linux Components," on page 15
- Section 2.2, "Using CASA with Linux," on page 18

For information on using CASA with Microsoft Windows, see Chapter 3, "CASA on Windows," on page 21.

2.1 Linux Components

As shown in Figure 1-1 on page 11, the main components of CASA on Linux are:

- Section 2.1.1, "CASA Identity Development Kit," on page 15
- Section 2.1.2, "miCASAd," on page 15
- Section 2.1.3, "Login Credential Capture Module," on page 15
- Section 2.1.4, "CASA Linux Packages," on page 16
- Section 2.1.5, "Linux Directories and Files," on page 16

2.1.1 CASA Identity Development Kit

Use the functions within this kit to write user/application credentials to the credential store. These functions internally store the credentials passed onto them by the applications in miCASAd. There are C, C++, C# and Java bindings available for the functions within this kit. See Section 2.1, "Linux Components," on page 15 and Section 3.1, "Windows Components," on page 21.

2.1.2 miCASAd

miCASAd is an active component that starts during boot time. It stores and provides credentials or secrets based on the Linux user identifier (uid) of the process that makes the IDK API calls. 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.

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.

2.1.3 Login Credential Capture Module

On Linux, the login credential capture module is implemented as a pluggable authentication module (PAM) (http://www.novell.com/documentation/oes/sles_admin/data/cha-pam.html). This PAM module captures the user's desktop login credentials and stores them in miCASAd using the IDK functions.

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.

IMPORTANT: Any PAM that uses the Identity Development Kit must temporarily set its effective user id 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. However, there might be cases where the user obtained by calling pam_get_user is not the user against whom the PAM module actually intends to store credentials.

2.1.4 CASA Linux Packages

CASA consists of two Linux packages:

- **CASA-1.5.*xxx*.i586.rpm:** Installs miCASAd, the startup scripts, the Login Credential Capture PAM module, and the relevant libraries required by any application that is using the CASA API.
- **CASA-devel-1.5.*xxx*.i586.rpm:** Installs the relevant header files that developers need to write applications to the CASA functions. This is dependent on CASA-1.5.*xxx*.i586.rpm.
- **CASA-gui-1.5.xxx.i586.rpm:** Installs CASA Manager which allows the end user to add, edit, and delete secrets. CASA Manager also allows the user to temporally suspend or lock the miCASA credential store.

All other directories are installed by CASA-1.5.0.i586.rpm, except /opt/novell/CASA/, which is the only directory installed by CASA-devel-1.5.0.i586.rpm.

2.1.5 Linux Directories and Files

CASA Linux files are located in the following directories:

- Section , "/opt/novell/CASA/doc," on page 16
- Section , "/opt/novell/CASA/lib or /opt/novell/CASA/lib64," on page 17
- Section , "/lib/security or /lib64/security," on page 17
- Section , "/opt/novell/CASA/bin," on page 17
- Section , "/opt/novell/CASA/images," on page 18
- Section , "/etc/init.d," on page 18
- Section , "/opt/novell/CASA/include," on page 18

/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.[version number]`	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®.
`Novell.CASA.Common.dll`	A common .NET library used by micasad and CASAManager.
`Novell.CASA.A-D.dll`	A .NET library that collects secrets from other credential stores.
`Novell.CASA.DataEngines.GnomeKeyRing.dll`	A C# wrapper to interact with GNOME Keyring.
`Novell.CASA.DataEngines.KWallet.dll`	A C# wrapper to interact with the KDE Wallet.
`Novell.CASAS.Policy.dll`	A .NET library to configures policy for miCASA.

/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.
`CASAManager.exe`	The management console used to view, edit, and delete secrets.
`CASAManager.sh`	The script file which starts CASAManager.

/opt/novell/CASA/images

The `/opt/novell/CASA/images` directory contains all images used by CASA Manager.

/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.There are links to this script from the appropriate runlevel 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.

2.2 Using CASA with Linux

- Section 2.2.1, “Linux Installation,” on page 18
- Section 2.2.2, “Starting, Stopping, and Restarting CASA on Linux,” on page 18
- Section 2.2.3, “Starting CASA Manager,” on page 19
- Section 2.2.4, “Linux Uninstallation,” on page 19

2.2.1 Linux Installation

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

On other distributions, use the following commands to install all of the required CASA components:

```
rpm -Uvh CASA-1.5.xxx.i586.rpm sdsd (CASA product installation)

rpm -Uvh CASA-devel-1.5.xxx.i586.rpm (CASA NDK installation)

rpm -Uvh CASA-gui-1.5.xxx.i586.rpm (CASA Manager installation)
```

2.2.2 Starting, Stopping, and Restarting CASA on Linux

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

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

2.2.3 Starting CASA Manager

Use the following command to start CASA Manager:

```
/opt/novell/CASA/bin/CASAManager.sh
```

2.2.4 Linux Uninstallation

Use the following commands to uninstall the CASA packages:

```
rpm  -e CASA-gui

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.

For information on using CASA with Linux*, see Chapter 2, "CASA on Linux," on page 15.

3.1 Windows Components

CASA consists of one Windows package, `CASA-1.5.0.msi`, which is the installation module that contains the following two components that match their Linux counterparts (see Section 2.1, "Linux Components," on page 15):

- `CASA-devel-1.5.0.msm`
- `CASA-1.5.0.msm`

3.1.1 Windows Directories and Files

CASA Windows files are located in the following directories:

\Program Files\Novell\CASA\bin

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

File	Description
`CASAManager.exe`	A management console for adding, editing, and deleting secrets.
`Lcredmgr.dll`	The login capture, login extension, and logout for Novell Client32.
`micasad.exe`	The miCASA service for Windows.
`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.
`jmicasa.dll`	The miCASA JNI library for the Java interface.

3.2 Using CASA with Windows

- Section 3.2.1, "Installing CASA on Windows," on page 23
- Section 3.2.2, "Starting CASA on Windows," on page 23

- Section 3.2.3, “Accessing CASA Manager,” on page 23
- Section 3.2.4, “Uninstalling CASA on Windows,” on page 24

3.2.1 Installing CASA on Windows

1. Before installing CASA on Windows, your system must be configured with the Microsoft .NET Framework and the Gtk# components that CASA requires. The CASA installation will determine if these software packages are already installed.
2. To install CASA on the Windows operating system, double-click the `CASA.msi` file.

3.2.2 Starting CASA on Windows

After installing CASA, you can start the Novell Identity Store service by either of the following methods:

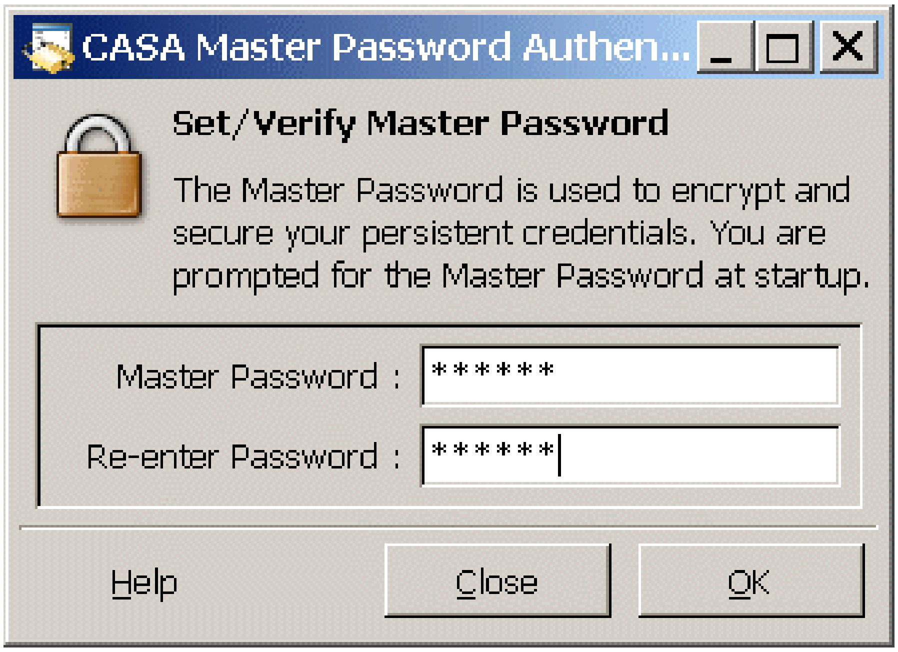
1. Click Start > Settings > Control Panel > Administrative Tools > Services > Novell Identity Store, or to start CASA automatically, reboot your machine.

3.2.3 Accessing CASA Manager

To run CASA Manager, double-click the CASA Manager icon on the desktop, or the `CASAManager.exe` file found in the `[Program files]\Novell\CASA\bin` directory.

1. The first time you run CASA Manager, you will be prompted to set a master password. This is used to encrypt and secure your persistent credentials. The Master Password must be at least eight characters in length.

Figure 3-1 *Set your master password when you start CASA Manager.*


CASA Master Password Authen...

Set/Verify Master Password

The Master Password is used to encrypt and secure your persistent credentials. You are prompted for the Master Password at startup.

Master Password : ******

Re-enter Password : ******

See also Chapter 4, “Administering CASA Manager,” on page 25.

3.2.4 Uninstalling CASA on Windows

To uninstall CASA, click *Start* > *Control Panel* > *Add/Remove Programs*, select CASA, then follow the instructions.

4 Administering CASA Manager

CASA Manager is the graphical user interface that enables you to access and manage the authentication credentials (secrets) of the programs and services installed on your Linux, Windows, or MacIntosh* devices.

WARNING: Because CASA collects and displays security credentials from secure applications running on your system, this software should not be used in any public environment where security might be compromised.

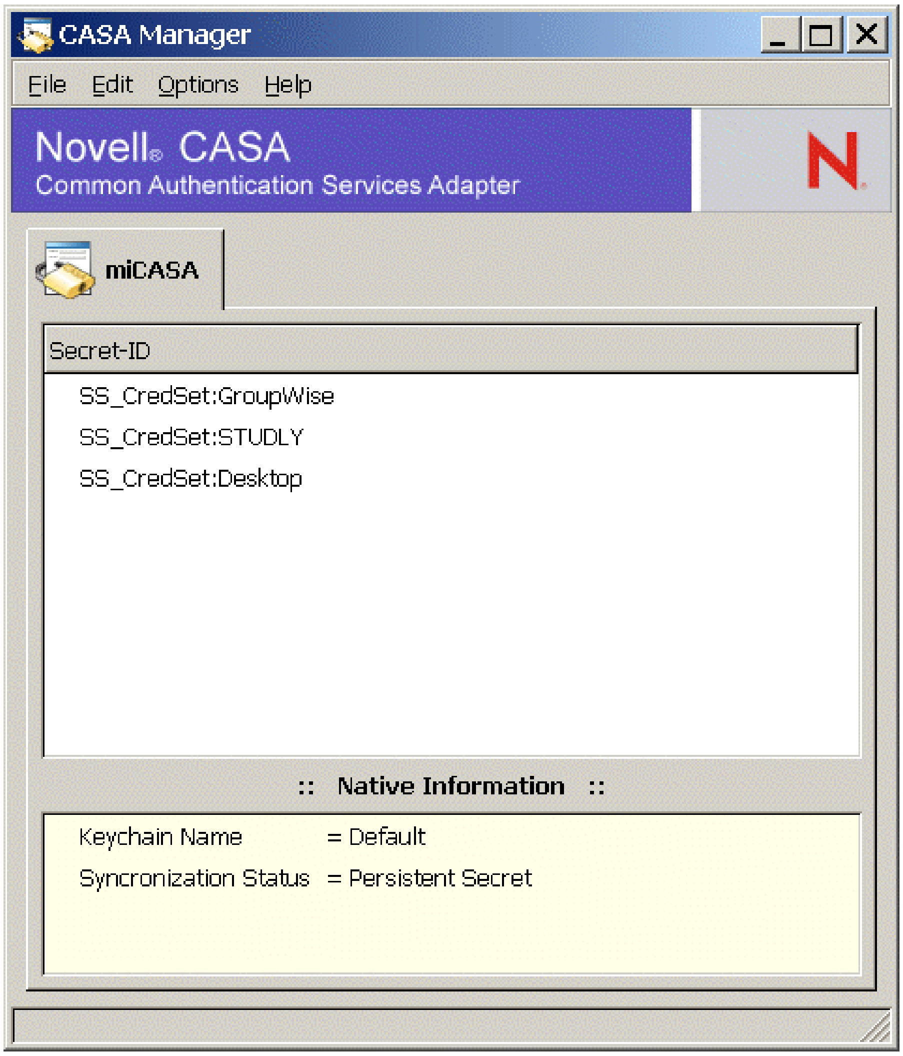
In addition, because CASA is integrated with your workstation login and other resident applications that require authentication credentials, you should create confidential passwords that are not easily broken to prevent unauthorized access.

To install CASA Manager on Linux, see Section 2.2, "Using CASA with Linux," on page 18. To install CASA Manager on Windows, see Section 3.2, "Using CASA with Windows," on page 22. CASA Manager has not yet been implemented for MacIntosh devices.

User credentials (secrets) are created automatically when installing and instantiating many routine applications and services on a system, such "name" and "password" values. For example, user secrets for SS_CredSet:GroupWise is created when the Novell® GroupWise® application is used, as shown in the Secrets-ID window in Figure 4-1 on page 26.

SS_CredSet identifies that a credential has one or more sets of key-value pairs assigned to it. The miCASA credential store is supported on Linux and Windows. CASA Manager also supports KDE Wallet and GNOME Keyring on Linux. CASA enables you to manage secrets among all three credential stores in Novell legacy applications or third-party applications.

Figure 4-1 CASA Manager GUI Showing Sample Credential Directory



SS_CredSet:GroupWise
SS_CredSet:STUDLY
SS_CredSet:Desktop

Keychain Name	= Default
Syncronization Status	= Persistent Secret

CASA Manager allows the user to view, edit, and add Secrets stored my the miCASA store. Applications such as Novell GroupWise, iPrint, and iFolder are CASA enabled and may store secrets in the miCASA store.

Secrets are stored in miCASA only in memory for CASA 1.0 and in an external directory in Version 1.5 or later. Session-based secrets imply 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 the workstation.

This section discusses the following topics:

- Section 4.1, "CASA Manager GUI Components," on page 27
- Section 4.2, "CASA Manager Functionality," on page 29
- Section 4.3, "Editing CASA Manager Options," on page 36

4.1 CASA Manager GUI Components

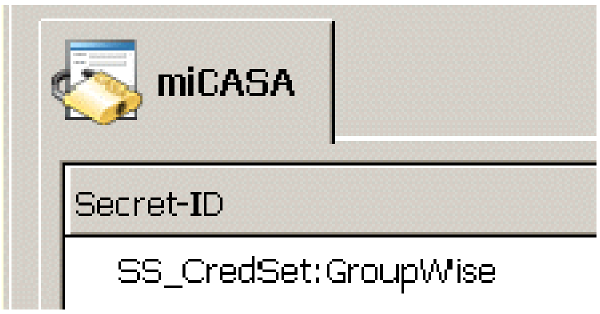
CASA Manager has the following components:

- Section 4.1.1, “Credential Store Tab,” on page 27
- Section 4.1.2, “Secret-ID Window,” on page 27
- Section 4.1.3, “Native Information Window,” on page 28

4.1.1 Credential Store Tab

In Figure 4-1 on page 26, the miCASA tab lists all secrets stored in the miCASA cache that CASA detects when CASA Manager is run. This example, which identifies three secrets cached on a Windows machine, displays only a single tab as shown below.

Figure 4-2 *CASA Manager Credential Store Tab*



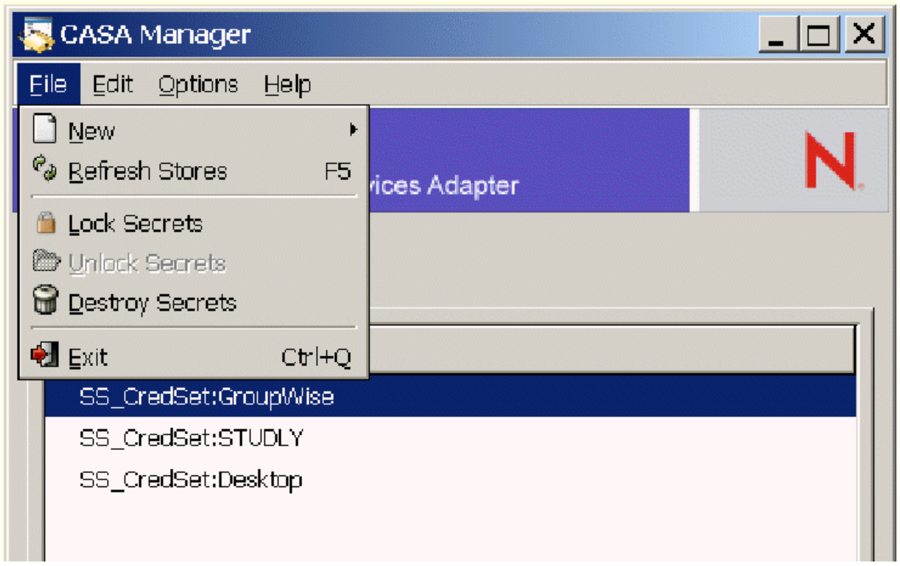
However, if CASA is installed on a Linux machine where KDE Wallet and GNOME Keyring are supported, for example, two additional tabs can be enabled to access all credentials cached in each of those credential stores. To access the secrets stored in each of these credential stores, you simply click on the individual tab.

NOTE: The example figure shows the tab only for the miCASA credential store.

4.1.2 Secret-ID Window

After selecting a credential store tab, the Secret-ID window displays the names of all secrets cached in the enabled credential store of your machine, as shown in the example in Figure 4-3 on page 28.

Figure 4-3 *Secret-ID Example*



SS_CredSet:GroupWise
SS_CredSet:STUDLY
SS_CredSet:Desktop

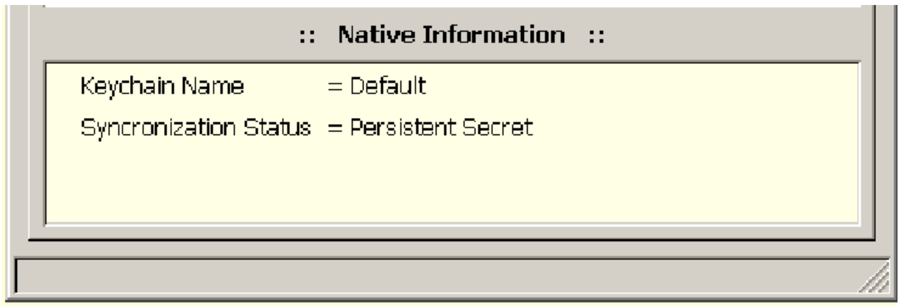
Select a secret to manage by either of two methods:

- Right-click the Secret-ID item listed in the window and select the task you wish to perform. This method allows you to do the following tasks:
    - Create new secrets
    - Create new keys
    - View and manage secrets and key-value pairs
    - Link keys and value pairs among secrets
    - Delete secrets stored in the session cache
- Click the Secret-ID item you wish to manage > Click one of the File/Edit/Options/Help functions in the menu.

4.1.3 Native Information Window

The Native Information window displays the attributes of the secrets that are cached in miCASA.

Figure 4-4 *Native Information Window*



:: Native Information ::

Keychain Name = Default
Syncronization Status = Persistent Secret

The Native Information window displays information about the secret. The information in this window will vary depending on which credential store is being viewed.

4.2 CASA Manager Functionality

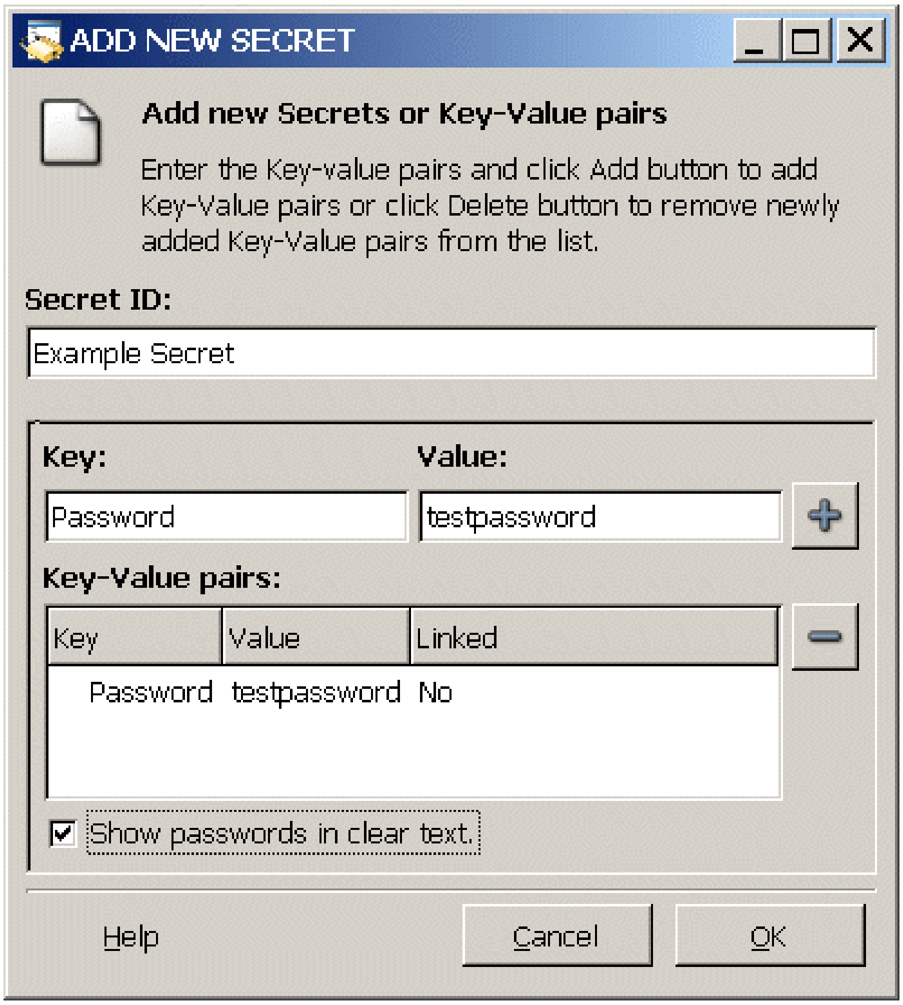
Secrets for each of the services shown in the Secret-ID window (Figure 4-1 on page 26), which are cached in miCASA, can be managed in the following ways:

- Section 4.2.1, "Creating Secrets," on page 29
- Section 4.2.2, "Refreshing Credential Stores," on page 31
- Section 4.2.3, "Locking Secrets," on page 31
- Section 4.2.4, "Destroying Secrets," on page 31
- Section 4.2.5, "Viewing Secret Values," on page 31
- Section 4.2.6, "Linking Secrets," on page 32
- Section 4.2.7, "Editing Secrets," on page 34
- Section 4.2.8, "Deleting Secrets," on page 35

4.2.1 Creating Secrets

CASA Manager enables you to manually create new secrets or to manage secrets that have been previously created by programs that integrate with CASA.

Figure 4-5 *Add New Secret or Key-Vale Pairs*



ADD NEW SECRET

Add new Secrets or Key-Value pairs

Enter the Key-value pairs and click Add button to add Key-Value pairs or click Delete button to remove newly added Key-Value pairs from the list.

Secret ID: Example Secret

Key: Password

Value: testpassword

Key-Value pairs:

Key	Value	Linked
Password	testpassword	No

[✓] Show passwords in clear text.

To manually create a new secret, use the following procedure:

1. In CASA Manager, click *File* > *New* > *New Secret*
2. Type the name that identifies the new secret in the Secret ID field, such as, Example Secret.
3. Type the name of the key and its value in the Key and Value fields, such as, Key: Password, and Value: testpassword.

   The asterisk (*) is the only restricted character in both the Key and Value fields.
4. Click the + button to add the newly formed Key -Value pair for the new secret.

   In the example shown in Figure 4-5 on page 30, the value of the password key is shown in clear text, that is “testpassword.” The password value is always be shown in encrypted form to help secure confidential information (that is, in asterisk characters) unless you select the *Show passwords in clear text*. You are then prompted to enter your master password to enable a single instance display of the password in the Value field.
5. Click *OK* to add the new secret, with its corresponding Key-Value pair, to the credential store.

   The secret now displays in the Secret-ID window, indicating that it has been added to the miCASA credential store.

4.2.2 Refreshing Credential Stores

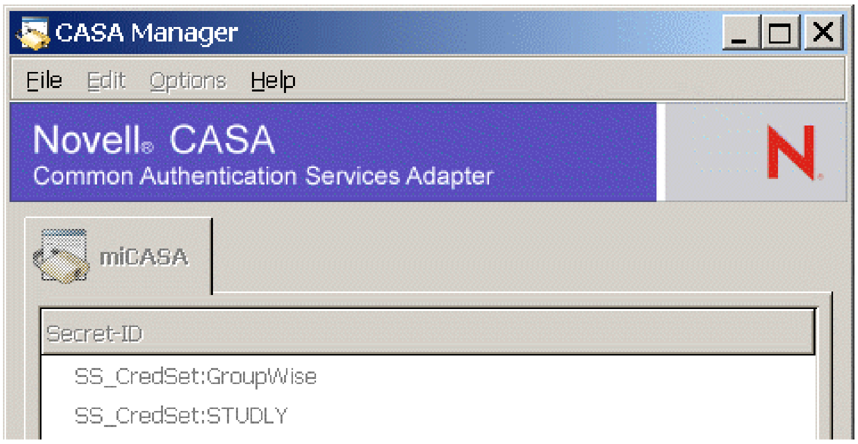
Refresh Stores menu option in CASA Manager is used to re-read all secrets in each of the configured stores. The miCASA credential store is supported on Linux and Windows. KDE Wallet and GNOME Keyring are additional stores supported on Linux..

4.2.3 Locking Secrets

To prevent individuals and other applications from viewing or manipulating your secrets, CASA Manager enables you to Lock Secrets. The Lock Secrets menu option temporarily disables the functionality of the miCASA stire. CASA-enabled applications are not able to read or write secrets to the miCASA store.

1 Click *File* > *Lock Secrets*.

Figure 4-6 *Locking CASA Manager*



Notice that all credential store tabs (miCASA, KDE Wallet, and GNOME Keyring) and cached secrets are dimmed when CASA is locked. Use the following procedure to unlock and restore functionality to CASA:

1a Click File > Unlock Secrets

1b Enter your master password

4.2.4 Destroying Secrets

Use the following procedure to clear your cache and destroy all credentials that are stored in memory:

1 Click *File* > *Destroy Secrets* > *OK*.

You can restore your secrets manually by creating new secrets or by using CASA-enabled applications to store your credentials in the miCASA store.

4.2.5 Viewing Secret Values

You can view the key-value pairs of all secrets cached in the miCASA credential store.

1 In the main Secret-ID window, click the secret you want to view.

2 Click *Edit* > *View* or press *F2*.

3 By default, key values are encrypted and displayed as asterisks. To show the value in clear text, click the “Show Values in clear text” box, and enter your master password.

4.2.6 Linking Secrets

You can link two or more secret keys so that their respective values are syncronized simultaneously. For example, you can link the CN of one secret to the password of another secret, all of the keys with one secret to each other, or any combination to synchronize all your secrets.

Currently, CASA only provides the ability to link keys within the miCASA credential store.

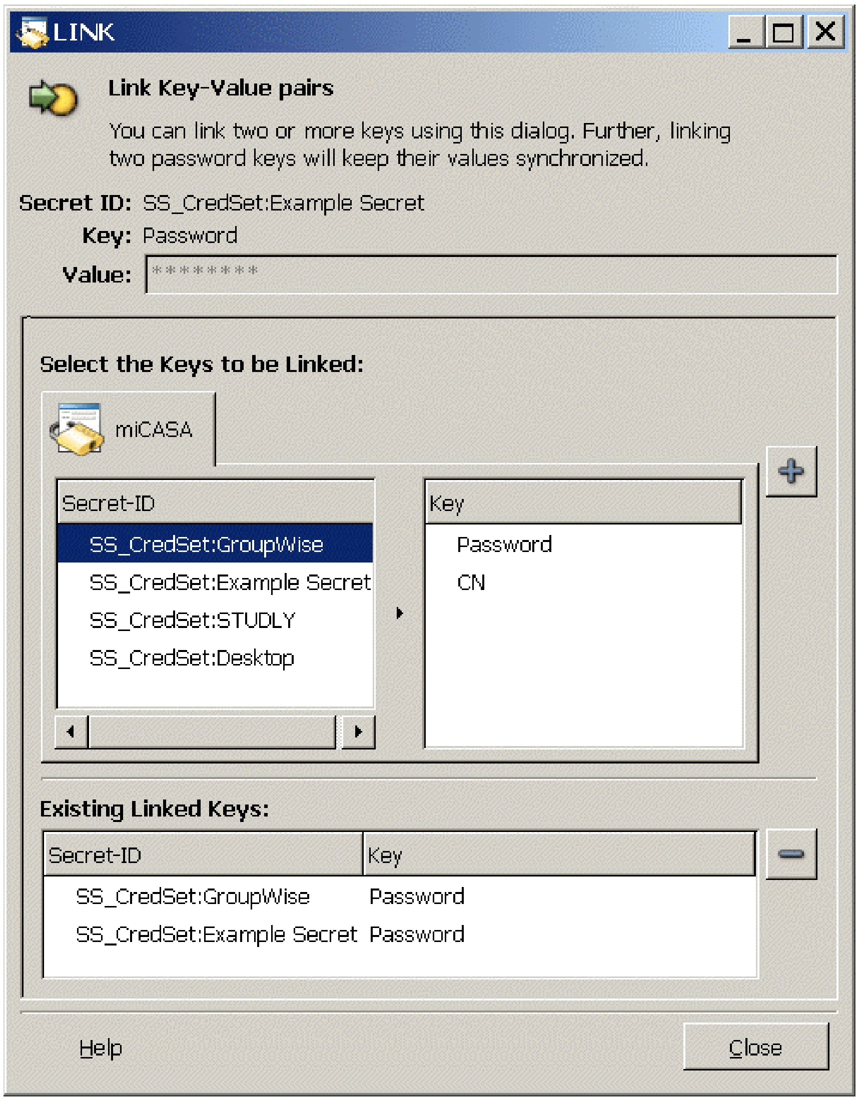
Link keys of secrets by using the Link feature from the CASA Manager:

1 Select the Secret you want to link and press F2 or select Edit > Link from the menu. This will open the Edit Secret and Key-Value pairs window.

2 Double-click on key in the Key field to open the Link management window.

  This utility enables you to link the key of any secret to the key of any other secret contained in the miCASA store.

Figure 4-7 *The CASA Link utility.*


LINK

Link Key-Value pairs

You can link two or more keys using this dialog. Further, linking two password keys will keep their values synchronized.

Secret ID: SS_CredSet:Example Secret
Key: Password
Value: ********

Select the Keys to be Linked:

miCASA

Secret-ID
SS_CredSet:GroupWise
SS_CredSet:Example Secret
SS_CredSet:STUDLY
SS_CredSet:Desktop

Key
Password
CN

Existing Linked Keys:

Secret-ID	Key
SS_CredSet:GroupWise	Password
SS_CredSet:Example Secret	Password

3 Click the Secret-ID you want to link. This will display all keys associated to this secret.

4 Click the Key you want to link, then click the + Button to link the selected key-value pair.

5 Repeat Step 4 to add and link as many secrets as you wish. All linked secrets and keys are displayed in the Existing Linked Keys window.

6 To verify if a secret is linked, view its status in the Edit Secret pairs window. The Link field displays either Yes or No. Verify by following any one of these steps:

- Double-click the secret.
- Right click the secret > click *View*.
- After selecting a Secret from the main window, press the F2 key.

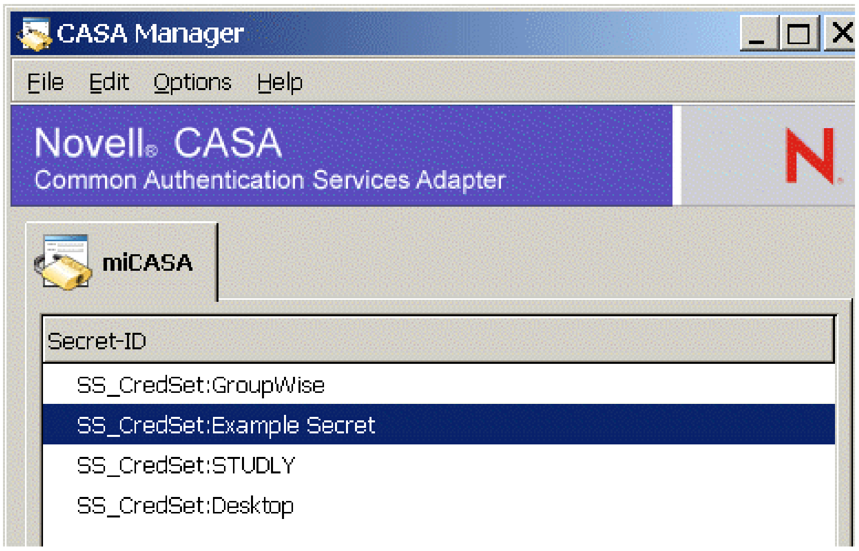
7 To unlink selected secrets, click any of the Secret-ID components listed in the Existing Linked Keys window, then click the – button. The selected secret is deleted from the Existing Linked Keys window.

4.2.7 Editing Secrets

NOTE: The Copy secrets feature is not available in CASA 1.5.

1 In the main Secret-ID window, double-click the secret you want to edit.

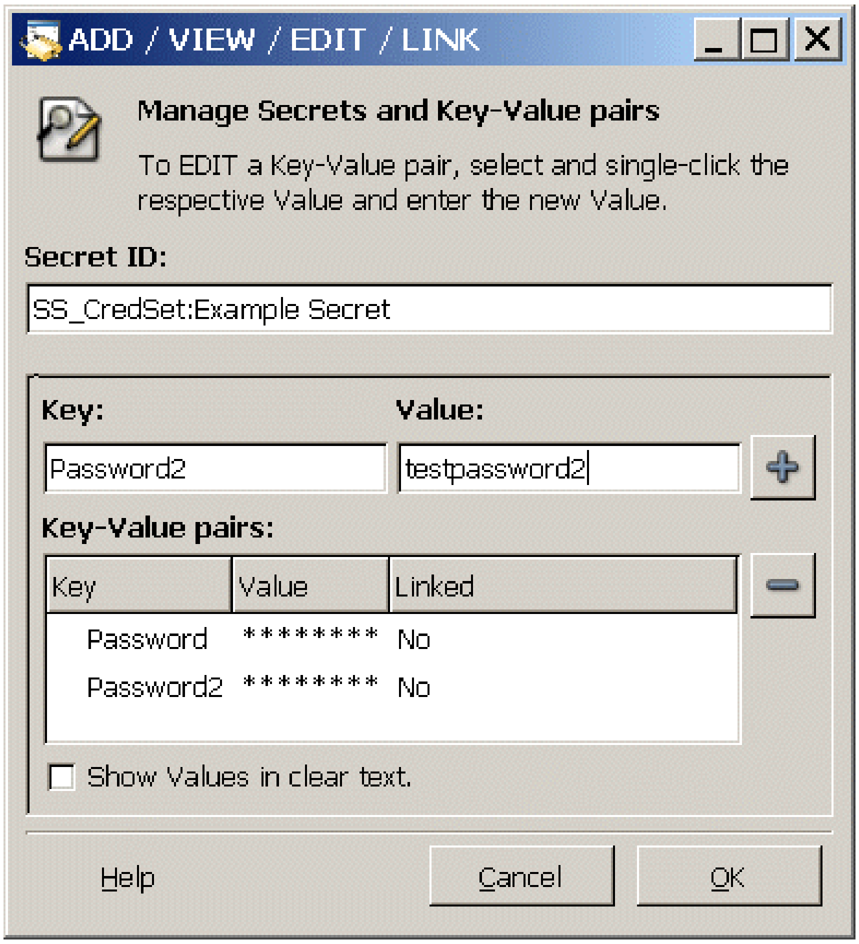
Figure 4-8 Select a secret to edit in CASA Manager.



SS_CredSet:GroupWise
SS_CredSet:Example Secret
SS_CredSet:STUDLY
SS_CredSet:Desktop

2 As shown below, you edit a secret by adding new or changing existing Key-Value pairs.

Figure 4-9 *CASA MAnager features an Add/View/Edit/Screen.*


ADD / VIEW / EDIT / LINK

Manage Secrets and Key-Value pairs

To EDIT a Key-Value pair, select and single-click the respective Value and enter the new Value.

Secret ID: SS_CredSet:Example Secret

Key: Password2

Value: testpassword2

Key-Value pairs:

Key	Value	Linked
Password	********	No
Password2	********	No

☐ Show Values in clear text.

In this example, a second password key and corresponding password value were added by typing “Password2” and “testpassword2” in the Key and Value fields, then clicking the + button.

In this example, the value is encrypted and displays as asterisks for the new Password2 key. To show the value in clear text, then click *Show Values in clear text*. You are then prompted to enter your master password before the values are displayed.

3 To edit the password value, click the Value field in the Key-Value pairs window, type your new value, then click *OK*. The new password value is saved in the miCASA credential store.

After they are created, Secret ID names cannot be edited.

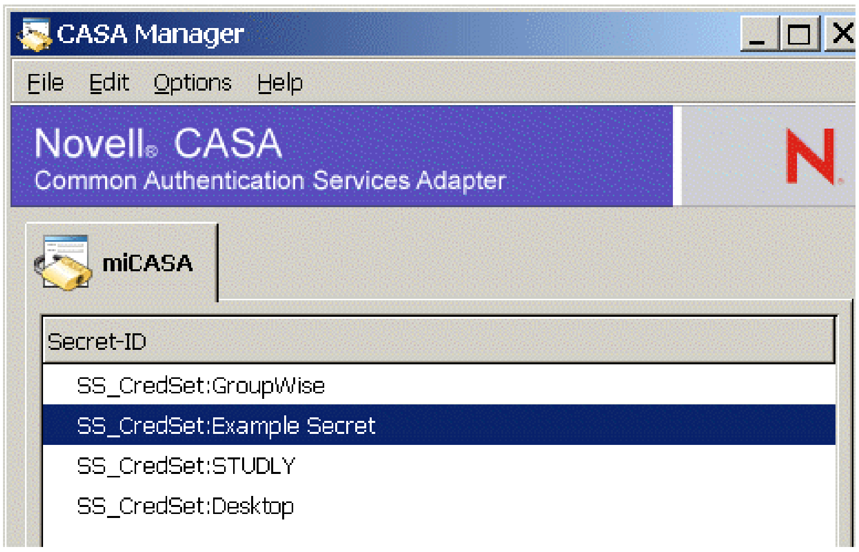
4.2.8 Deleting Secrets

Use the following procedure to delete a secret from the credential store:

1 In the main Secret-ID window, right-click the secret you want to delete, then click *Delete*.

Alternatively, select *Edit* > *Delete* in the main menu.

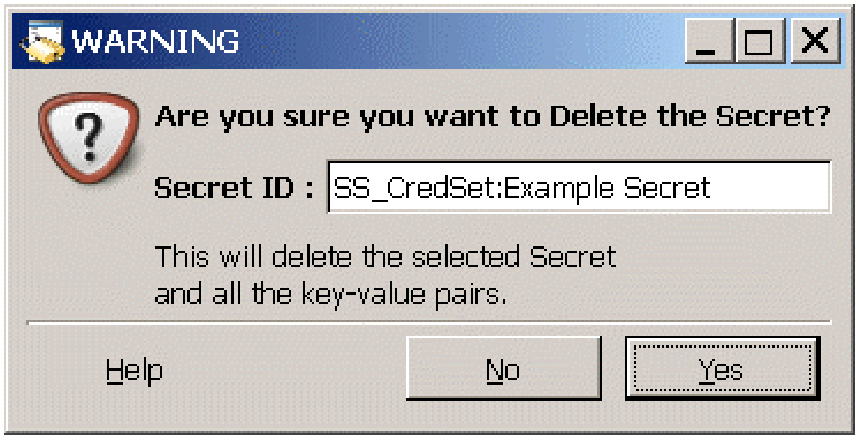
Figure 4-10 Edit a secret by selecting it from CASA Manager.



SS_CredSet:GroupWise
SS_CredSet:Example Secret
SS_CredSet:STUDLY
SS_CredSet:Desktop

2 Click Yes to delete the selected secret and all of its associated key-value pairs.

Figure 4-11 Deleting secrets also deletes all of its associated key-value pairs.



Are you sure you want to Delete the Secret?

Secret ID : SS_CredSet:Example Secret

This will delete the selected Secret and all the key-value pairs.

4.3 Editing CASA Manager Options

- Section 4.3.1, "Setting CASA Preferences," on page 36
- Section 4.3.2, "Setting Persistent Storage," on page 37

4.3.1 Setting CASA Preferences

This option is active only when CASA is installed in a Linux environment. CASA Manager provides support for miCASA, KDE Wallet, and GNOME Keyring credential stores. Use the following procedure to specify which credential stores you want to use:

1 From the CASA Manager page, click *Options* > *Preferences*.

2 The miCASA store is always active and cannot be removed. You can select additional credential stores you wish to use in CASA Manager (that is, KDE Wallet or GNOME Keyring).

NOTE: In Windows, miCASA is the only credential store available, so references to KDE Wallet and GNOME Keyring are inactive.

4.3.2 Setting Persistent Storage

CASA automatically saves your secrets on your computer and retrieves them the next time you login. Your secrets are encrypted using the password used for login, as well as the master password required to use CASA Manager. When the desktop password changes, you must enter your master password to decrypt your saved secrets.

To change your master password:

1. Enter your old master password, enter your new master password twice, then click *OK*.

Your master password must be at least eight characters long.

Functions

5

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

- "miCASAGetCredential" on page 40
- "miCASARemoveCredential" on page 42
- "miCASASetCredential" on page 43

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 located in the default install directory.

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 or 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. Supported types are:

Value	Description
SSCS_CRED_TYPE_BASIC_F	***

credential

(OUT) Points to the credential structure SSCS_BASIC_CREDENTIAL (page 46).

ext

Reserved for future use.

Return Values

If successful, returns one of the following:

- The credential for the sharedSecretID, if one is requested, and found.
- The credential for the appSecretID, if the sharedSecretID is not found or not requested.

- The default credential if Steps 1 and 2 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 unique string that represents the name of the credential that should be removed, such as Novell.GroupWise or Novell.iFolder.

sharedSecretID

(IN) Ignored for this release.

ext

Reserved for future use.

Return Values

If successful, returns 0. Otherwise, returns a non-zero error code.

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 or 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.

Return Values

If successful, set a credential and returns 0.

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.

Structures

6

CASA uses the following structures:

- "SSCS_BASIC_CREDENTIAL" on page 46
- "SSCS_SECRET_ID_T" on page 47

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). Currently, `0` is the only support flag.

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 either the secret or the credential.

Revision History

A

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

November 18, 2005	• Updated CASA documentation and deliverables from version 1.0 to 1.5. • Documented new CASA Manager functionality in Chapter 4, "Administering CASA Manager," on page 25.
October 5, 2005	Transitioned to revised Novell® documentation standards.
June 15, 2005	Revised documentation to coincide with Version 1.0 software updates.
June 3, 2005	Posted as beta documentation.