

## Administration Guide

# Novell® Common Authentication Services Adapter (CASA)

**1.7**

April 05, 2011

[www.novell.com](http://www.novell.com)



## Legal Notices

Novell, Inc. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc. makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. See the [Novell International Trade Services Web page \(http://www.novell.com/info/exports/\)](http://www.novell.com/info/exports/) for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Copyright © 1993-2011 Novell, Inc. All rights reserved. No part of this publication may be reproduced, photocopied, stored on a retrieval system, or transmitted without the express written consent of the publisher.

Novell, Inc.  
404 Wyman Street, Suite 500  
Waltham, MA 02451  
U.S.A.  
[www.novell.com](http://www.novell.com)

*Online Documentation:* To access the latest online documentation for this and other Novell products, see [the Novell Documentation Web page \(http://www.novell.com/documentation\)](http://www.novell.com/documentation).

## Novell Trademarks

For Novell trademarks, see [the Novell Trademark and Service Mark list \(http://www.novell.com/company/legal/trademarks/tmlist.html\)](http://www.novell.com/company/legal/trademarks/tmlist.html).

## Third-Party Materials

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

Novell Modular Authentication Services (NMAS) software includes support for a number of login methods from third-party authentication developers. Refer to the NMAS Partners Web site (<http://www.novell.com/products/nmas/partners/>) for a list of authorized NMAS partners and a description of their login methods.

Each NMAS partner addresses network authentication with unique product features and characteristics. Therefore, each login method will vary in its actual security properties. Novell has not evaluated the security methodologies of these partner products, and while these products may have qualified for the Novell Yes, Tested and Approved or Novell Directory Enabled logos, those logos only relate to general product interoperability. Novell encourages you to carefully investigate each NMAS partner's product features to determine which product will best meet your security needs. Also, some login methods require additional hardware and software not included with the NMAS product.

# Contents

<b>About This Guide</b>	<b>5</b>
<b>1 Getting Started</b>	<b>7</b>
1.1 Credentials	8
1.2 Sharing Credentials	8
<b>2 CASA on Linux</b>	<b>11</b>
2.1 Linux Components	11
2.1.1 CASA Identity Development Kit	11
2.1.2 miCASA	11
2.1.3 Login Credential Capture Module	11
2.1.4 CASA Linux Packages	12
2.1.5 Linux Directories and Files	12
2.2 Using CASA with Linux	14
2.2.1 Installing CASA on Linux	15
2.2.2 Configuring CASA on Linux	15
2.2.3 Starting, Stopping, and Restarting CASA on Linux	15
2.2.4 Uninstalling CASA	16
2.2.5 Accessing CASA Manager: Linux	16
2.2.6 Avoiding Conflicting Service Realms	16
2.2.7 Using CASA in chroot Environments	17
<b>3 CASA on Windows</b>	<b>19</b>
3.1 Windows Components	19
3.1.1 Windows Directories and Files	19
3.2 Using CASA with Windows	20
3.2.1 Installing CASA on Windows	20
3.2.2 Starting CASA on Windows	20
3.2.3 Accessing CASA Manager	20
3.2.4 Uninstalling CASA on Windows	21
<b>4 CASA Manager Administration</b>	<b>23</b>
4.1 CASA Manager GUI Components	24
4.1.1 Credential Store Tab	25
4.1.2 Secret-ID Window	25
4.1.3 Native Information Window	26
4.2 Editing CASA Manager Options	27
4.2.1 Assigning Single Sign-on Preferences	27
4.2.2 Setting Persistent Storage	30
4.2.3 Changing Your Master Password	30
4.2.4 Setting CASA Preferences	31
4.2.5 Creating Secret Policies	32
4.3 CASA Manager Functionality	33
4.3.1 Creating Secrets	33
4.3.2 Refreshing Credential Stores	34
4.3.3 Locking and Unlocking Secrets	34
4.3.4 Destroying Secrets	35

4.3.5	Exporting User Secrets . . . . .	35
4.3.6	Importing User Secrets . . . . .	37
4.3.7	Viewing Secret Values . . . . .	37
4.3.8	Linking Secrets . . . . .	38
4.3.9	Copying Secrets . . . . .	39
4.3.10	Finding and Replacing Secrets . . . . .	42
4.3.11	Editing Secrets . . . . .	45
4.3.12	Deleting Secrets . . . . .	46
4.4	Resetting the CASA Master Password . . . . .	47
<b>5</b>	<b>Functions</b>	<b>51</b>
	miCASAGetCredential . . . . .	51
	miCASAOpenSecretStoreCache . . . . .	52
	miCASAReadBinaryKey . . . . .	53
	miCASAReadKey . . . . .	54
	miCASARemoveCredential . . . . .	55
	miCASARemoveKey . . . . .	56
	miCASASetCredential . . . . .	57
	miCASAWriteBinaryKey . . . . .	58
	miCASAWriteKey . . . . .	60
<b>6</b>	<b>Structures</b>	<b>63</b>
	SSCS_BASIC_CREDENTIAL . . . . .	63
	SSCS_SECRET_ID_T . . . . .	63
<b>A</b>	<b>CASA Error Codes</b>	<b>65</b>
<b>B</b>	<b>CASA Security Guidelines</b>	<b>69</b>
B.1	CASA Security Administration . . . . .	69
B.2	CASA User Security . . . . .	70
B.3	Security Considerations for CASA Developers . . . . .	71
<b>C</b>	<b>Revision History</b>	<b>73</b>

# About This Guide

The Common Authentication Service Adapter (CASA) 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:

- ♦ [Chapter 1, “Getting Started,” on page 7](#)
- ♦ [Chapter 2, “CASA on Linux,” on page 11](#)
- ♦ [Chapter 3, “CASA on Windows,” on page 19](#)
- ♦ [Chapter 4, “CASA Manager Administration,” on page 23](#)
- ♦ [Chapter 5, “Functions,” on page 51](#)
- ♦ [Chapter 6, “Structures,” on page 63](#)
- ♦ [Appendix C, “Revision History,” on page 73](#)

## 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 understand 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 documentation, visit the [Novell Common Authentication Service Adapter Web site \(http://forge.novell.com/modules/xfmod/project/?casa\)](http://forge.novell.com/modules/xfmod/project/?casa).

## Additional Documentation

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



# Getting Started

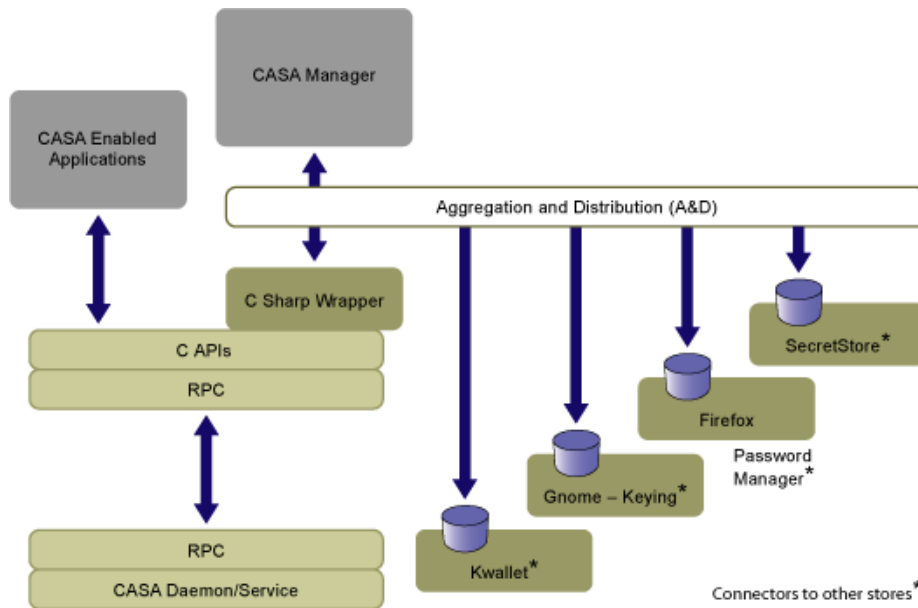
# 1

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 1.7 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

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.

- ♦ [Section 1.1, “Credentials,” on page 8](#)
- ♦ [Section 1.2, “Sharing Credentials,” on page 8](#)

## 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. 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, which is defined by the network administrator, can be an eDirectory tree, an Active Directory domain, a managed database, or even a combination of all of these. Multiple applications commonly authenticate to the same realm and the miCASA functions enable applications to share authentication 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 guide 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.

---



This section contains information on using the Common Authentication Service Adapter (CASA) developer kit on Linux.

- ♦ [Section 2.1, “Linux Components,” on page 11](#)
- ♦ [Section 2.2, “Using CASA with Linux,” on page 14](#)

For information on using CASA with Microsoft Windows, see [Chapter 3, “CASA on Windows,” on page 19](#).

## 2.1 Linux Components

The main components of CASA on Linux are:

- ♦ [Section 2.1.1, “CASA Identity Development Kit,” on page 11](#)
- ♦ [Section 2.1.2, “miCASA,” on page 11](#)
- ♦ [Section 2.1.3, “Login Credential Capture Module,” on page 11](#)
- ♦ [Section 2.1.4, “CASA Linux Packages,” on page 12](#)
- ♦ [Section 2.1.5, “Linux Directories and Files,” on page 12](#)

### 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 to them by the applications in miCASA. There are C, C++, C# and Java bindings available for the functions within this kit. See [Section 2.1, “Linux Components,” on page 11](#) and [Section 3.1, “Windows Components,” on page 19](#).

### 2.1.2 miCASA

miCASA is an active CASA 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, miCASA 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.

Credentials are encrypted by the desktop password and stored on the user’s file system. When the user logs back into the workstation, the encrypted credentials are decrypted and reloaded in memory. Applications can then use these credentials to authenticate to additional services.

### 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](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 miCASA 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 miCASA and in the session stack, then close the user's session with miCASA.

---

**IMPORTANT:** Any PAM module 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 the following Linux packages:

- ♦ **CASA-1.7.xxx.i586.rpm:** Installs miCASA, 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-1.7.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-1.7.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.
- ♦ **CASA-kwallet-1.7.xxx.i586.rpm:** Enables the CASA GUI to perform Aggregation and Distribution tasks for managing KWallet through CASA in KDE desktop.
- ♦ **yast2-CASA-1.7.xxx.i586.rpm:** Assists administrators to configure and enable CASA through YaST after system installation.

## 2.1.5 Linux Directories and Files

CASA Linux files are located in the following directories:

- ♦ ["/usr/share/doc/packages/CASA/doc" on page 12](#)
- ♦ ["/usr/lib or /usr/lib64" on page 13](#)
- ♦ ["/lib/security or /lib64/security" on page 13](#)
- ♦ ["/usr/bin" on page 13](#)
- ♦ ["/usr/share/doc/packages/CASA/images" on page 14](#)
- ♦ ["/etc/init.d" on page 14](#)
- ♦ ["/usr/include" on page 14](#)
- ♦ ["/usr/share/omc/svcinfo.d" on page 14](#)

### **/usr/share/doc/packages/CASA/doc**

This directory contains the following files:

File	Description
<code>casa_enu.pdf</code>	This document.

File	Description
Readme.txt	The Readme file, which contains information about any last-minute updates.

### **/usr/lib or /usr/lib64**

This directory contains the following files for 32-bit machines (/usr/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 miCASA 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.CASA.CASAPolicy.dll	A .NET library to configure the policies 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.

### **/usr/bin**

This directory contains the following files:

File	Description
micasad.exe	The miCASA daemon that starts at runlevels 1, 2, 3, and 5. It is based on Mono.
micasad-init	A script that starts micasad.exe.
CASAManager.exe	The management console used to view, edit, and delete secrets.

File	Description
CASAManager.sh	The script file that starts CASAManager.

## **/usr/share/doc/packages/CASA/images**

This directory contains all images used by CASA Manager.

## **/etc/init.d**

This directory contains the following file:

File	Description
micasad	The micasad startup script. This script is started in runlevels 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 /usr/sbin/micasad.sh script to start the daemon.

## **/usr/include**

This 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.
casa_status.h	The header file for the CASA auth-token functions.
micasa_types.h	The common header file defining supported types.
sscs_string.h	The header file that contains common string operations.
sscs_utf8.h	The header file for common UTF8 string operations.

## **/usr/share/omc/svcinfo.d**

This directory contains the following file:

File	Description
micasad.xml	The service description XML file.

## **2.2 Using CASA with Linux**

- ♦ [Section 2.2.1, “Installing CASA on Linux,” on page 15](#)
- ♦ [Section 2.2.2, “Configuring CASA on Linux,” on page 15](#)
- ♦ [Section 2.2.3, “Starting, Stopping, and Restarting CASA on Linux,” on page 15](#)
- ♦ [Section 2.2.4, “Uninstalling CASA,” on page 16](#)
- ♦ [Section 2.2.5, “Accessing CASA Manager: Linux,” on page 16](#)

- ♦ [Section 2.2.6, “Avoiding Conflicting Service Realms,” on page 16](#)
- ♦ [Section 2.2.7, “Using CASA in chroot Environments,” on page 17](#)

## 2.2.1 Installing CASA on Linux

Review the following sections:

- ♦ [“YaSt Installation” on page 15](#)
- ♦ [“Command Line Installation” on page 15](#)

### YaSt Installation

- 1 Log in to the Linux device as the `root` user.
- 2 Insert the Linux installation media.
- 3 Run Yast to open the YaST Control Center.
- 4 Click *Software > Software Management*.
- 5 In the *Search* option, specify CASA and click *OK* to list all the CASA packages.
- 6 Select all the listed CASA packages and click *Install > Apply*.

### Command Line Installation

CASA is preinstalled on the Novell Linux Desktop operating system.

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

```
rpm -Uvh CASA-1.7.xxx.i586.rpm (CASA product installation)
rpm -Uvh CASA-devel-1.7.xxx.i586.rpm (CASA NDK installation)
rpm -Uvh CASA-gui-1.7.xxx.i586.rpm (CASA Manager installation)
rpm -Uvh CASA-kwallet-1.7.xxx.i586.rpm
rpm -Uvh yast2-CASA-1.7.xxx.i586.rpm
```

## 2.2.2 Configuring CASA on Linux

- 1 Log in to the Linux device as the `root` user.
- 2 Run Yast to open the YaST Control Center.
- 3 Click *Security and Users*.
- 4 Click *CASA*.
- 5 In the CASA Configuration screen, click *Enable CASA*, then click *Finish*.
- 6 Restart the device.

## 2.2.3 Starting, Stopping, and Restarting CASA on Linux

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

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

## 2.2.4 Uninstalling CASA

Use the following commands to uninstall the CASA packages:

```
rpm -e CASA-gui
rpm -e CASA
rpm -e CASA-kwallet
rpm -e yast2-CASA
```

## 2.2.5 Accessing CASA Manager: Linux

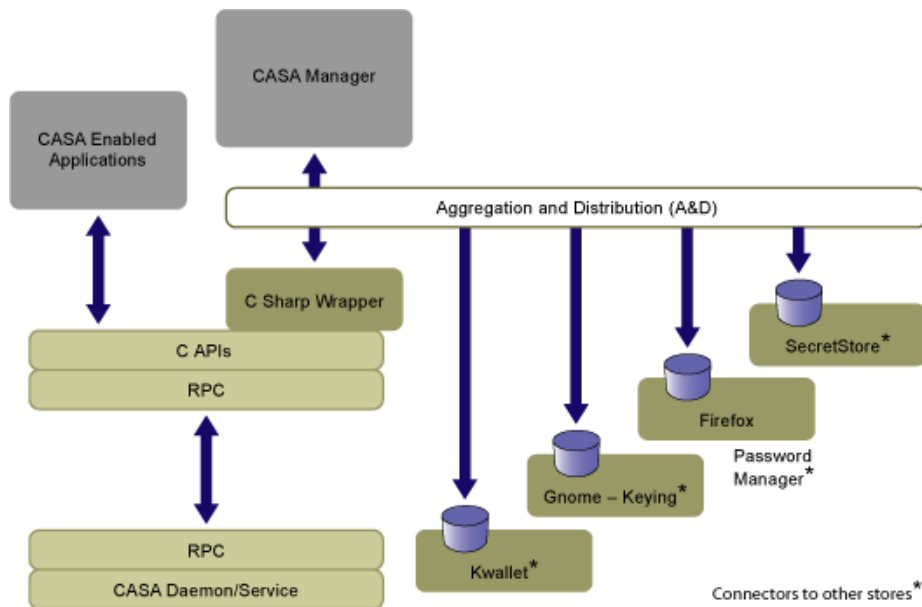
- 1 Run the following command:

```
CASAManager.sh
```

- 2 In the Password screen that pops up, 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.

The CASA Manager applications starts up.

**Figure 2-1** From the Novell CASA Manager application, you administer all secrets associated with the current user.



See also [Chapter 4, “CASA Manager Administration,”](#) on page 23.

## 2.2.6 Avoiding Conflicting Service Realms

Currently, the YaST module for configuring the CASA auth-token service allows multiple realms to be added with the same realm name, which can lead to confusion. For example, if you add two realms both named “adam” with different configurations, both realms appear in YaST and you cannot differentiate between them. Moreover, when you delete one of the “adam” realms, the configuration information for the remaining realm named “adam” also is deleted.



To prevent this conflict, make sure to uniquely name any realm added to the service when you configure CASA.

## 2.2.7 Using CASA in chroot Environments

For an application to access CASA in a change root (chroot) environment, the root user must do the following:

- 1** Copy the micasa libraries to the chroot directory, as shown in the following example:  

```
cp /usr/lib/libmicasa* <chrootdir>/usr/lib/
```
- 2** Create the CASA socket file in the chroot directory by linking the current CASA socket. An example of how to do this is explained below:
  - 2a** From the <chrootdir>/var/run directory, execute the following command:  

```
ln /var/run/.novellCASA
```

---

**NOTE:** The chrootdir must be on the same disk partition as /var/run.

---



This section contains information on using the Common Authentication Service Adapter (CASA) on Microsoft Windows.

- ♦ [Section 3.1, “Windows Components,” on page 19](#)
- ♦ [Section 3.2, “Using CASA with Windows,” on page 20](#)

For information on using CASA with Linux, see [Chapter 2, “CASA on Linux,” on page 11](#).

## 3.1 Windows Components

CASA consists of one Windows package, `CASA.msi`, which is the installation module that contains the following components that match their Linux counterparts (see [Section 2.1, “Linux Components,” on page 11](#)):

- ♦ `CASA-gui.msm`
- ♦ `CASA.msm`

A separate Windows package called `CASA-devel.msi` installs the CASA development kit.

### 3.1.1 Windows Directories and Files

CASA Windows files are located in the following directories:

- ♦ [“\Program Files\Novell\CASA\bin” on page 19](#)
- ♦ [“\Program Files\Novell\CASA\doc” on page 19](#)
- ♦ [“\windows\system32\(64\)” on page 20](#)

#### **\Program Files\Novell\CASA\bin**

The directory contains the following files:

File	Description
<code>CASAManager.exe</code>	A management console for adding, editing, and deleting secrets.
<code>lcredmgr.dll</code>	The login capture, login extension, and logout for Novell Client32.
<code>miCASA.exe</code>	The micasad service for Windows.
<code>Novell.CASA.miCASAWrapper.dll</code>	The miCASA C# developer kit library, which is based on .NET.
Other supporting libraries	See linux section for complete list of files.

#### **\Program Files\Novell\CASA\doc**

This 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)**

This 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 20](#)
- ♦ [Section 3.2.2, “Starting CASA on Windows,” on page 20](#)
- ♦ [Section 3.2.3, “Accessing CASA Manager,” on page 20](#)
- ♦ [Section 3.2.4, “Uninstalling CASA on Windows,” on page 21](#)

### **3.2.1 Installing CASA on Windows**

- 1 Before installing CASA on Windows, make sure your system is configured with the Microsoft.NET Framework and the Gtk# components that CASA requires (see [“Windows Components” on page 19](#)). 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, which is located in the CASA installation directory.

### **3.2.2 Starting CASA on Windows**

After installing CASA, ensure that the Novell Identity Store service is running. To start the service:

- 1 Run `services.msc` to launch the Services Window.
- 2 Right-click *Novell Identity Store* and click *Start*.

### **3.2.3 Accessing CASA Manager**

To run CASA Manager:

- 1 In the Windows *Start* menu, click *Programs > Novell CASA*.

- 2 (Optional) Run the `CASA Launcher .exe` file found in the `[Program files]\Novell\CASA\bin` directory.
- 3 The first time you run CASA Manager, you are 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.



See also [Chapter 4, “CASA Manager Administration,”](#) on page 23.

### 3.2.4 Uninstalling CASA on Windows

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



# CASA Manager Administration

# 4

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

---

**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.

For more information about enhancing security when using CASA, see [Appendix B, “CASA Security Guidelines,” on page 69](#).

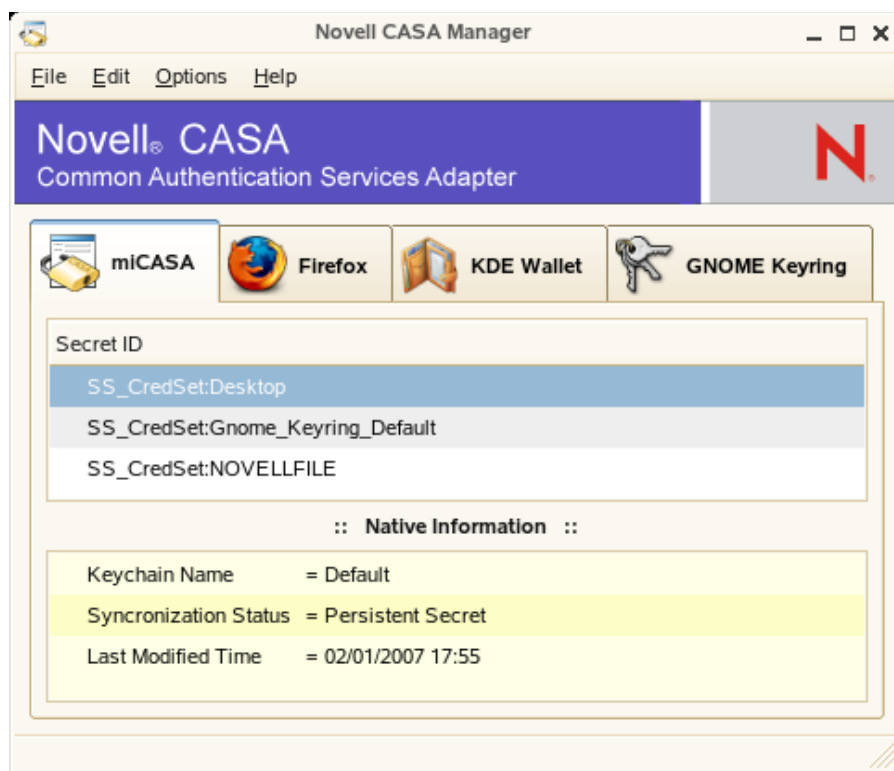
---

To install CASA Manager on Linux, see [Chapter 2, “CASA on Linux,” on page 11](#). To install CASA Manager on Windows, see [Section 3.2, “Using CASA with Windows,” on page 20](#).

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 are created when the Novell GroupWise application is used, as shown in the Secrets-ID window in [Figure 4-1 on page 24](#).

SS\_CredSet identifies that a credential has one or more sets of key-value pairs assigned to it. The miCASA credential store and Firefox Password Manager store are supported on Linux and Windows. CASA Manager also supports KDE Wallet and GNOME Keyring on Linux. CASA Manager enables you to manage secrets in other credential stores or third-party applications (such as the Firefox application, as shown in [Figure 4-1](#)).

**Figure 4-1** CASA Manager GUI Showing a Sample Credential Directory and Third-party Application



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

For CASA 1.0, secrets are stored in miCASA only in memory. In CASA 1.5 or later versions, secrets are stored in miCASA and persistently on the file system. Through configuration, secrets can be tagged as session-based or non-persistent.

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 the 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 24](#)
- ♦ [Section 4.2, “Editing CASA Manager Options,” on page 27](#)
- ♦ [Section 4.3, “CASA Manager Functionality,” on page 33](#)
- ♦ [Section 4.4, “Resetting the CASA Master Password,” on page 47](#)

## 4.1 CASA Manager GUI Components

CASA Manager has the following components:

- ♦ [Section 4.1.1, “Credential Store Tab,” on page 25](#)

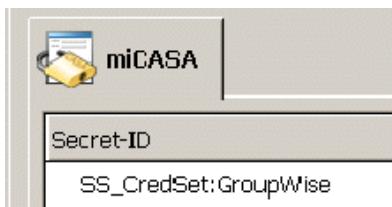


- ♦ [Section 4.1.2, “Secret-ID Window,” on page 25](#)
- ♦ [Section 4.1.3, “Native Information Window,” on page 26](#)

### 4.1.1 Credential Store Tab

In [Figure 4-1 on page 24](#), the miCASA tab lists all secrets stored in the miCASA cache when CASA Manager is run. The following example identifies three secrets cached on a Windows machine:

**Figure 4-2** CASA Manager Credential Store Tab



However, suppose that CASA is installed on a Linux machine where KDE Wallet, GNOME Keyring, and Firefox credential stores are supported. As shown in [Figure 4-3 on page 25](#), three additional tabs are enabled to access all credentials cached in each of those stores. To access the secrets stored in each of these credential stores, you simply click the individual tab.

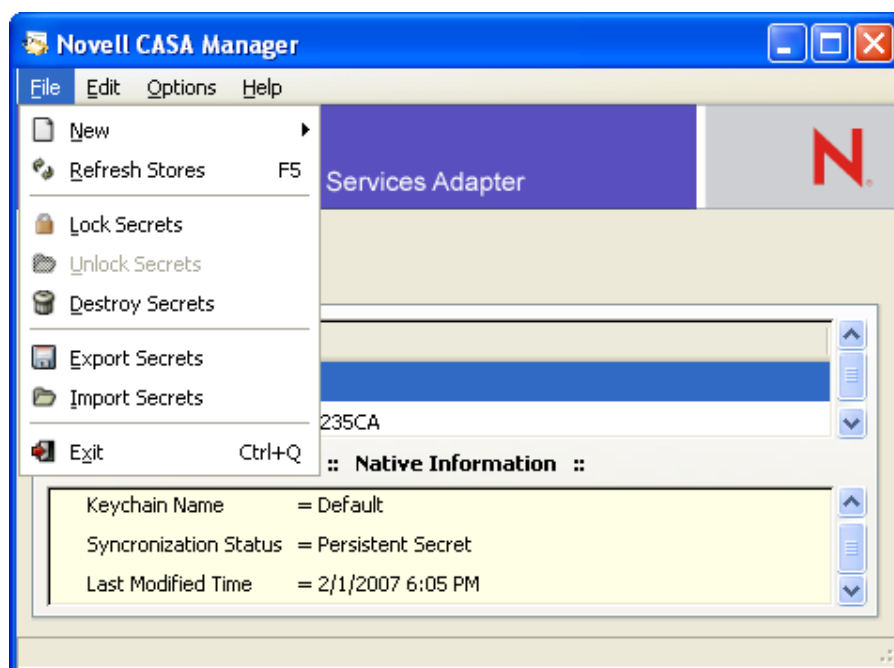
**Figure 4-3** In Linux, navigate between enabled credential stores by clicking on their individual tabs.



### 4.1.2 Secret-ID Window

After you select 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 4-4 on page 26](#).

**Figure 4-4** Secret-ID Example



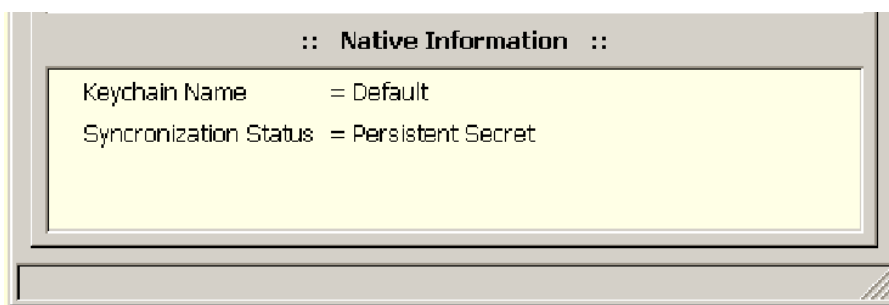
You can select a secret to manage by either of two methods:

- ♦ Right-click the Secret-ID item listed in the window and select the task you want 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
  - ♦ Copy the selected secrets to a location you select
  - ♦ Delete secrets stored in the session cache
- ♦ Click the Secret-ID item you want to manage, then 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 and information for the secret that is currently selected.

**Figure 4-5** Native Information Window



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 Editing CASA Manager Options

- ♦ [Section 4.2.1, “Assigning Single Sign-on Preferences,” on page 27](#)
- ♦ [Section 4.2.2, “Setting Persistent Storage,” on page 30](#)
- ♦ [Section 4.2.3, “Changing Your Master Password,” on page 30](#)
- ♦ [Section 4.2.4, “Setting CASA Preferences,” on page 31](#)
- ♦ [Section 4.2.5, “Creating Secret Policies,” on page 32](#)

### 4.2.1 Assigning Single Sign-on Preferences

On Linux, CASA Manager enables you to configure and store user sign-on information for the following services:

- ♦ [“Konquerer Web Services” on page 27](#)
- ♦ [“CASA Network Manager” on page 29](#)

#### Konquerer Web Services

The CASA Konquerer Web utility allows you to select a preconfigured CASA template to help you provide the user credentials required to access a particular service:

- ♦ A generic Web site that you designate
- ♦ Any number of Novell Web sites
- ♦ Gmail
- ♦ Yahoo! Mail
- ♦ MSN Hotmail

After configuring a template, the user’s sign-in credentials are automatically entered into the *username* and *password* fields, thus providing a seamless single sign-on user experience. When configuring the CASA template, the KDE wallet subsystem might prompt you to allow CASA Manager to access its storage system.

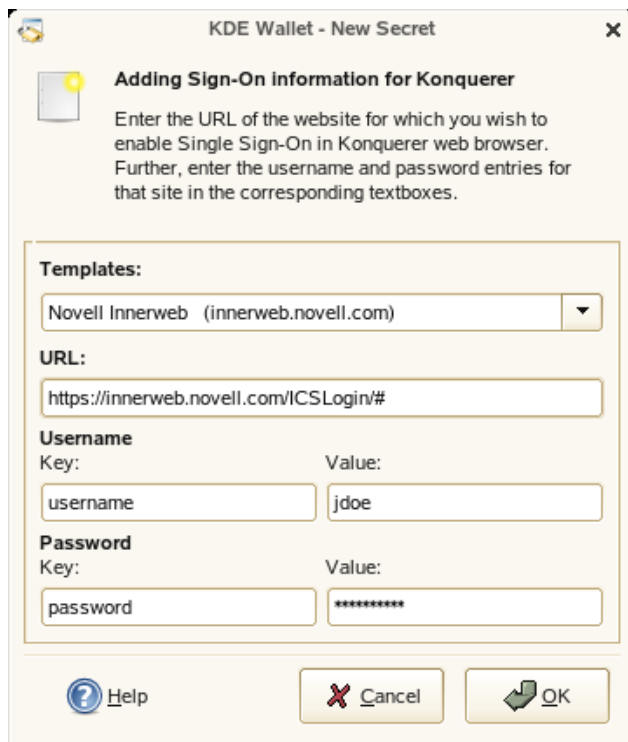
Here's an example how to configure single sign-on for a user accessing a Novell Innerweb account:

- 1 In CASA Manager, click *Options > Application SSO > Konquerer*.

The KDE Wallet wizard opens and prompts you for user information for a Web service.

- 2 In the drop-down Templates menu, click *Novell Innerweb (innerweb.novell.com)*, then fill in the values for the *username* and *password*.

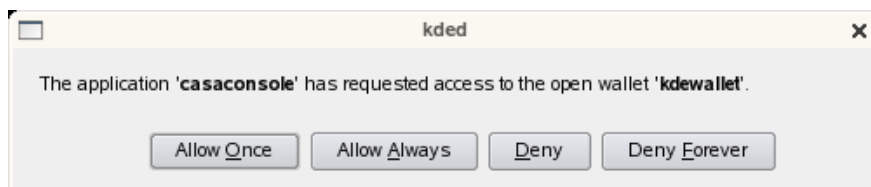
The template automatically fills in the proper login URL for this Web site:



The image shows a KDE Wallet dialog box titled "KDE Wallet - New Secret". It contains a section "Adding Sign-On information for Konquerer" with instructions to enter a URL, username, and password. The "Templates:" dropdown is set to "Novell Innerweb (innerweb.novell.com)". The "URL:" field contains "https://innerweb.novell.com/ICSLogin/#". The "Username" section has a "Key:" field with "username" and a "Value:" field with "jdoe". The "Password" section has a "Key:" field with "password" and a "Value:" field with "\*\*\*\*\*". At the bottom are buttons for "Help", "Cancel", and "OK".

- 3 Click *OK*.

- 4 The following dialog might appear indicating that CASA Manager is attempting to modify the contents of the KDE Wallet.

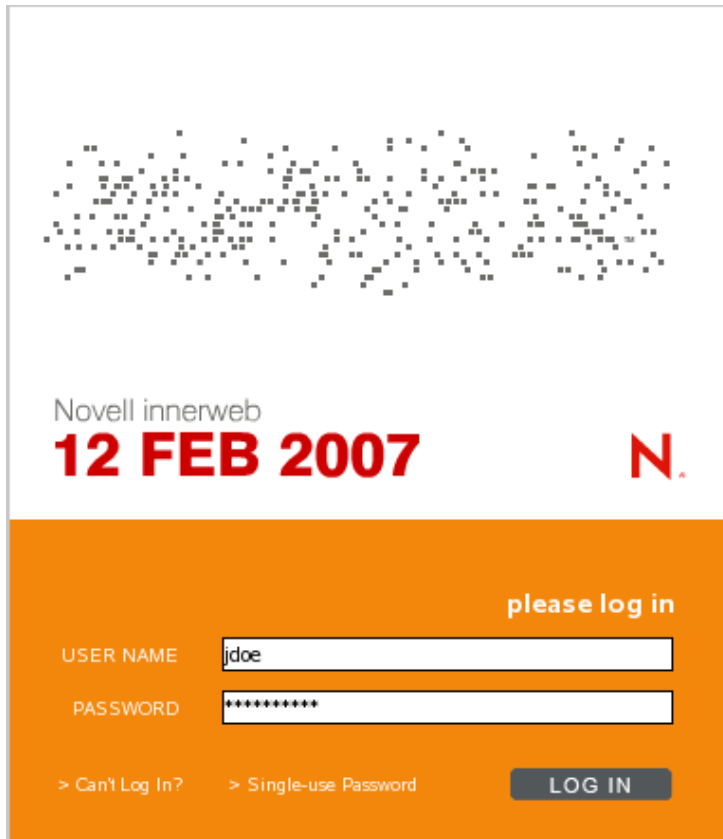


The image shows a kded dialog box with the title "kded". It contains the text "The application 'casaconsole' has requested access to the open wallet 'kdewallet'." Below the text are four buttons: "Allow Once", "Allow Always", "Deny", and "Deny Forever".

To allow CASA Manager to interact with KDE wallet, select *Allow Once* or *Allow Always*.

This setting can be changed later by repeating this procedure for the Web page and changing your preference.

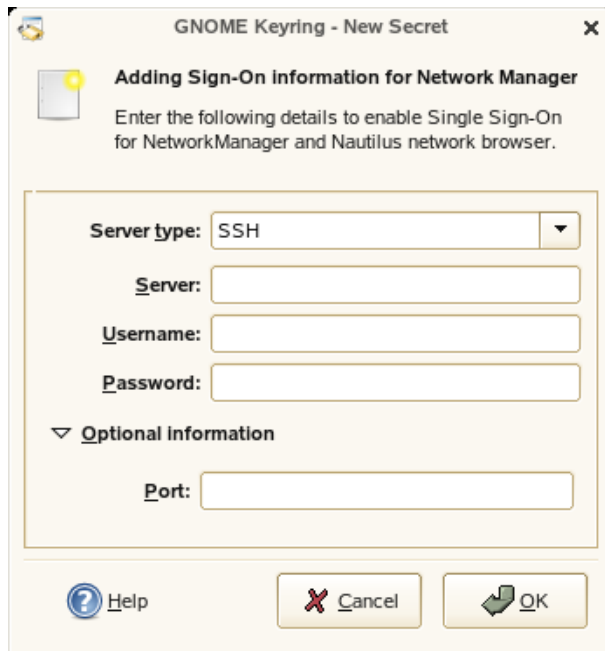
- 5 Open a browser and enter the URL of the Web page you configured in [Step 2](#). The *user name* and *password* are automatically filled in, if you allowed this option in the previous step.

A screenshot of a web login page for Novell innerweb. The page has a white header area with a pixelated graphic of a city skyline. Below the graphic, the text "Novell innerweb" is displayed in a small, grey font. To the right of this text is a large, red "N" logo. In the center of the header, the date "12 FEB 2007" is shown in a large, bold, red font. The main body of the page is orange. In the top right corner of the orange area, the text "please log in" is written in a small, white font. Below this, there are two input fields. The first is labeled "USER NAME" in a small, white font, and it contains the text "jdoe". The second is labeled "PASSWORD" in a small, white font, and it contains a series of asterisks. Below the input fields, there are two links: "> Can't Log In?" and "> Single-use Password", both in a small, white font. To the right of these links is a dark grey button with the text "LOG IN" in a small, white font.

## CASA Network Manager

The CASA NetworkManager utility enables single sign-on to the Linux NetworkManager and the Nautilus network browser using the following procedure:

- 1 In CASA Manager, click *Options > Application SSO > NetworkManager*. The GNOME Keyring window opens.



- 2 Enter the appropriate server and user information details required to establish a single sign-on network connection:

**Server type:** The type of server connection. Specify SSH, FTP (with login), or Windows Share.

**Server:** The name of the server. Specify the fully designated name of the server to which you want to connect.

**Username:** Enter the name of the authorized user accessing the network.

**Password:** Enter the user's password.

**Port:** Specify the assigned port number to access the network.

- 3 Click *OK*.

The user is now provisioned to access the specified server using single sign-on.

## 4.2.2 Setting Persistent Storage

CASA automatically stores your secrets on your computer's hard drive and retrieves them the next time you log in. Your secrets are encrypted using the password you 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 retrieve your saved secrets. This occurs when you run CASAManager the first time after booting the computer.

## 4.2.3 Changing Your Master Password

To change your master password:

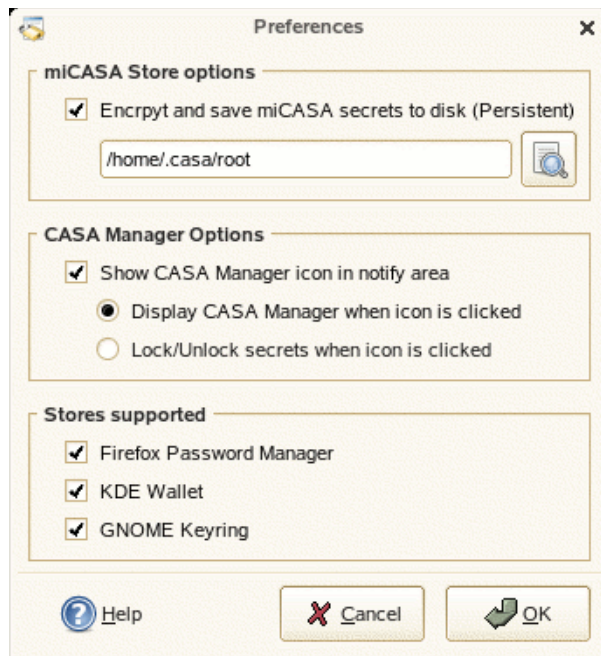
- 1 Click *Options* > select *Change Master Password*.
- 2 Enter your old master password, enter your new master password twice, then click *OK*.

Your master password must be at least eight characters long.

## 4.2.4 Setting CASA Preferences

Manage CASA preferences using the following procedure:

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



- 2 Select the CASA preference settings you desire.

With CASA 1.7, you can select where miCASA stores credentials persistently. By default, your secrets are stored in the following locations:

- ♦ **Linux:** /home/.casa/[username]
- ♦ **Windows:** /Documents and Settings/casa

You can configure whether or not to show the CASA icon in the task bar. If selected, you can also configure the action when you click the icon. The default action is to display CASA Manager when the icon is clicked.

You can also change that action to lock and unlock the miCASA store. This prevents applications from reading and writing secrets temporarily, until unlocked or when the workstation is restarted.

CASA Manager supports miCASA, Firefox, KDE Wallet (Linux only), and GNOME Keyring (Linux only) credential stores. The three preference options are:

- ♦ miCASA
- ♦ KDE Wallet
- ♦ GNOME Keyring

On Windows, you can only enable or disable the Firefox Password Manager.

The miCASA store is always active and cannot be removed. You can select additional credential stores to use in CASA Manager (that is, Firefox Password Manager, KDE Wallet, GNOME Keyring, or other credential stores that are available to CASA Manager).

## 4.2.5 Creating Secret Policies

After creating user secrets in the credential stores, CASA Manager enables you to quickly designate whether or not those secrets are stored as persistent or non-persistent secrets.

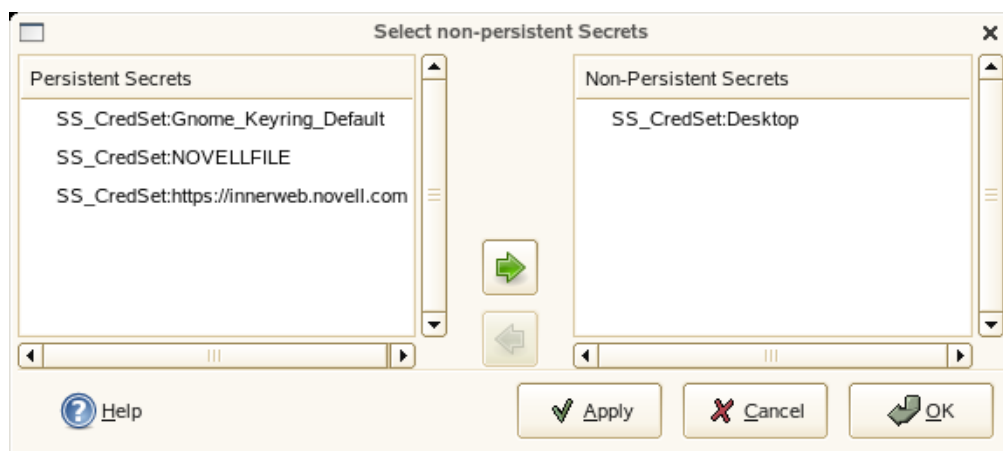
Persistent secrets are miCASA credentials that are stored on the file system and available for authenticating to Web and network services after restarting the workstation. Non-persistent secrets are not stored on the file system and are removed when a machine is shut down or CASA is restarted. Use the following procedure to change the persistence status of user secrets:

- 1 In CASA Manager, click *Options > Policies*, or press the *F3* key.

**Figure 4-6** All of the user's secrets are displayed in a two-panel window.



- 2 Double-click each secret you want to change from a persistent to non-persistent policy, or you can click a secret, then click the green arrow to change the persistence of the secret.



- 3 Click *Apply* to accept and save the policy, then click *OK*. The designated persistence policy is applied immediately and will remain until you change it using this procedure.



## 4.3 CASA Manager Functionality

Secrets for each of the services shown in the Secret-ID window (Figure 4-1 on page 24), which are cached in miCASA, can be managed in the following ways:

- ♦ [Section 4.3.1, “Creating Secrets,” on page 33](#)
- ♦ [Section 4.3.2, “Refreshing Credential Stores,” on page 34](#)
- ♦ [Section 4.3.3, “Locking and Unlocking Secrets,” on page 34](#)
- ♦ [Section 4.3.4, “Destroying Secrets,” on page 35](#)
- ♦ [Section 4.3.5, “Exporting User Secrets,” on page 35](#)
- ♦ [Section 4.3.6, “Importing User Secrets,” on page 37](#)
- ♦ [Section 4.3.7, “Viewing Secret Values,” on page 37](#)
- ♦ [Section 4.3.8, “Linking Secrets,” on page 38](#)
- ♦ [Section 4.3.9, “Copying Secrets,” on page 39](#)
- ♦ [Section 4.3.10, “Finding and Replacing Secrets,” on page 42](#)
- ♦ [Section 4.3.11, “Editing Secrets,” on page 45](#)
- ♦ [Section 4.3.12, “Deleting Secrets,” on page 46](#)

### 4.3.1 Creating Secrets

To manually create a new secret:

- 1 In CASA Manager, click *File > New > New Secret*.

**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:

Help Cancel OK

2 Fill in the following fields:

- ♦ **Secret ID:** Type the name that identifies the new secret such as, *Example Secret*.
- ♦ **Key:** Type the name of the key such as *Password*.
- ♦ **Value:** Type the name of the key's value such as *testpassword*.

3 Click the + button to add the newly formed key-value pair for the new secret.

In the example shown in [Step 1 on page 33](#), the value of the password key is shown in clear text, that is “testpassword.” The password value is always shown in encrypted form (that is, in asterisk characters) to help secure confidential information unless you select *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.

4 Click *OK* to add the new secret, with its corresponding key-value pairs, to the credential store.

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

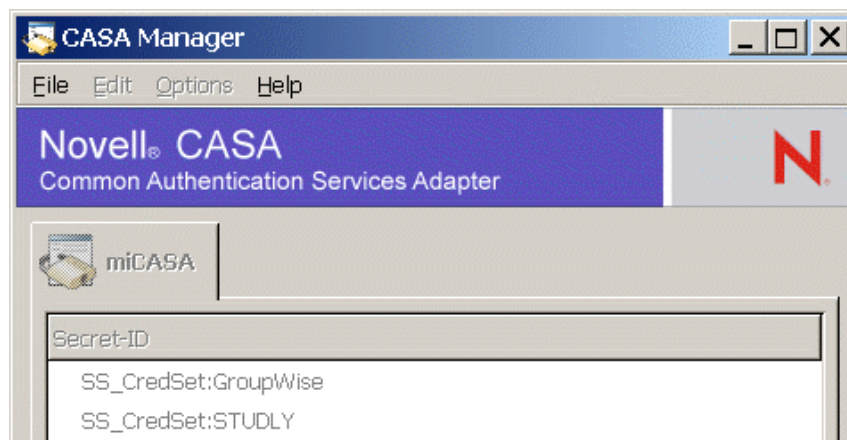
## 4.3.2 Refreshing Credential Stores

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

## 4.3.3 Locking and Unlocking 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 store, so CASA-enabled applications cannot read or write secrets to the miCASA store.

1 Click *File > Lock Secrets*.



All credential store tabs (miCASA, Firefox, KDE Wallet, and GNOME Keyring) and cached secrets are dimmed when CASA is locked.

To unlock and restore functionality to CASA:

- 1 Click *File > Unlock Secrets*.
- 2 Enter your master password.

### 4.3.4 Destroying Secrets

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

- 1 Click the tab for the credential store you want to clear.



- 2 Click *File > Destroy Secrets > OK*.
  - 2a (Option) In the Secret ID window, right-click on the credential you want to clear > click *Delete* > click *OK*.

You can restore your secrets manually by [creating new secrets](#) or by using CASA Manager or CASA-enabled applications to store your credentials in the miCASA store.

### 4.3.5 Exporting User Secrets

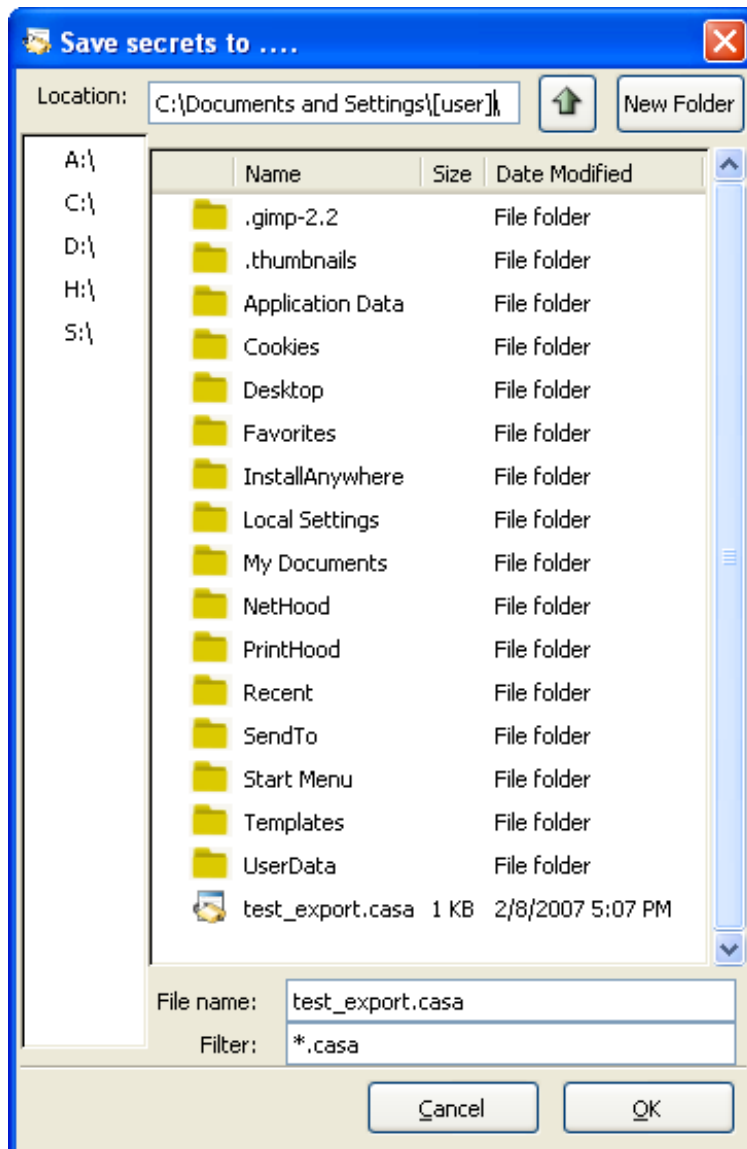
You can export your miCASA secrets to a specific directory you select, either with or without encryption, using the following procedure:

- 1 Click *File > Export Secrets*.

The *CASA - Export Secrets* window prompts you for your master password and whether or not to encrypt the secrets when they are exported. To enhance security, secrets are encrypted by default and you must check *Do not encrypt export file* to export secrets in clear text.



- 2 Select a directory in which to export your miCASA secrets, enter a file name, then click *OK*.



In this example, the secrets are exported to the Desktop with the name of `test_export`.

---

**NOTE:** External storage devices, such as portable USB drives, also can be used to store exported secrets. It is strongly recommended that secrets always be encrypted when stored on any external storage device.

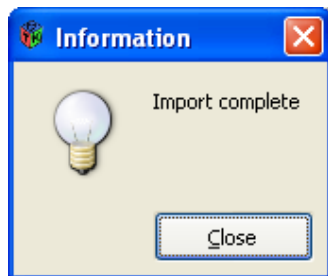
---

### 4.3.6 Importing User Secrets

You can import user secrets from a file where they have been previously saved using the following procedure:

- 1 Click *File > Import Secrets*. CASA Manager opens the Save Secrets window, listing all directories available to the user.
- 2 Navigate to the directory where the user secrets were previously exported, as explained in [Section 4.3.5, “Exporting User Secrets,” on page 35](#). These stored files are identified with the `.casa` extension in the file name.
- 3 Click the file (in this example, `test_export.casa`) > *OK*.
- 4 If requested, enter the master password used to encrypt this file, then click *OK*.

User secrets are then imported to and the information advisory window displays *Import Complete*.



- 5 Click *Close* to return to CASA Manager.

### 4.3.7 Viewing Secret Values

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

- 1 In CASA Manager, click the tab of the credential store you want to view.
- 2 In the main Secret-ID window, click the secret you want to view.
- 3 Click *Edit > View* or press *F2*.
- 4 By default, key values are encrypted and displayed as asterisks. To show the value in clear text, click *Show Values in clear text* and enter your master password.

### 4.3.8 Linking Secrets

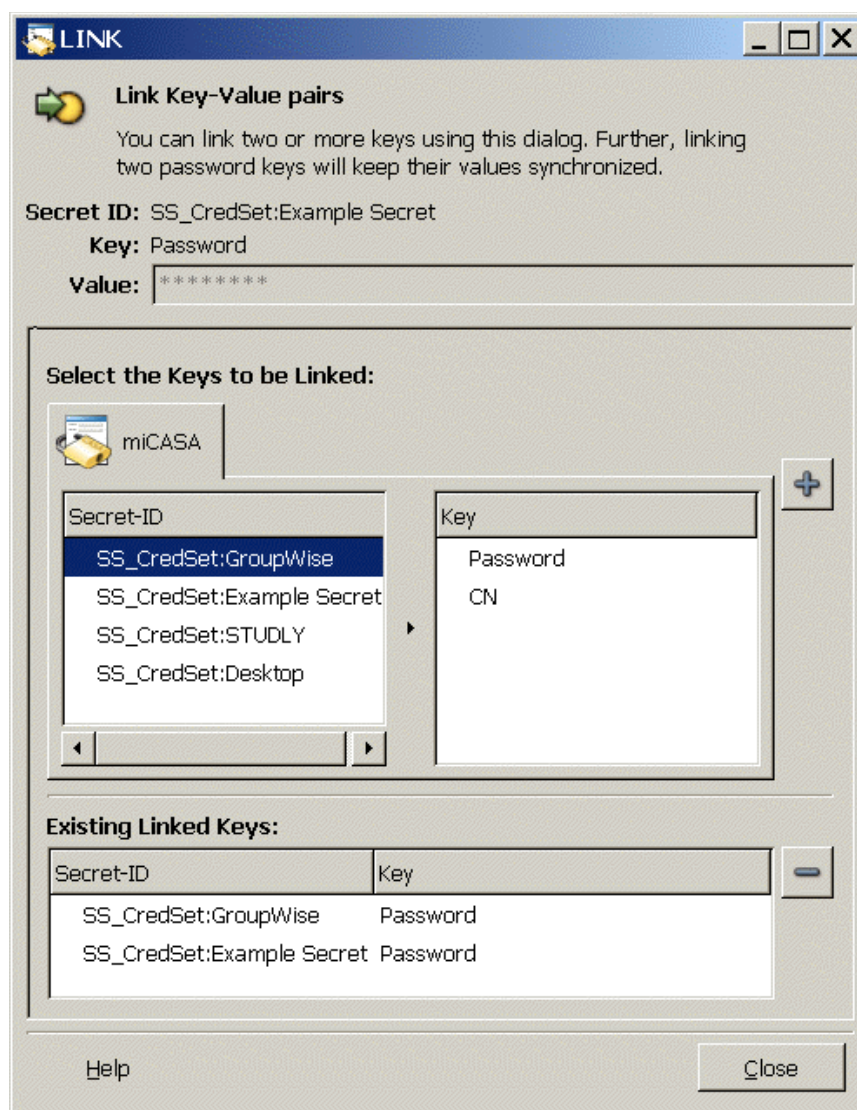
You can link two or more secret keys so that their respective values are synchronized. For example, you can link the CN of one secret to the password of another secret, all of the keys with one secret to each other, or any combination to synchronize all your secrets.

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

To link secrets:

- 1 Select the secret you want to link, then press F2 or select *Edit > Link*. This opens the Edit Secret and Key-Value Pairs window.
- 2 Double-click a key in the *Key* field to open the Link management window.

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



- 3 Click the Secret-ID you want to link. This displays 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 want.

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 by following any one of these steps:

- ♦ Double-click the secret.
- ♦ Right-click the secret, then click *View*.
- ♦ Select a secret from the main window, then press the F2 key.

The Link field displays either Yes or No.

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

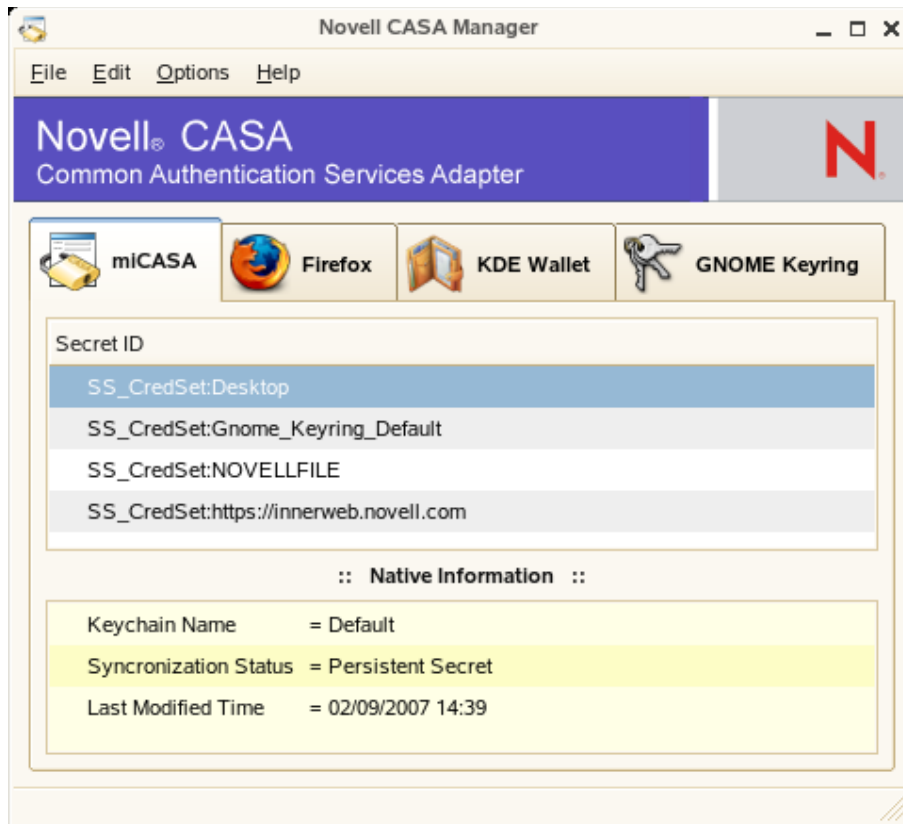
The selected secret is deleted from the Existing Linked Keys window.

### 4.3.9 Copying Secrets

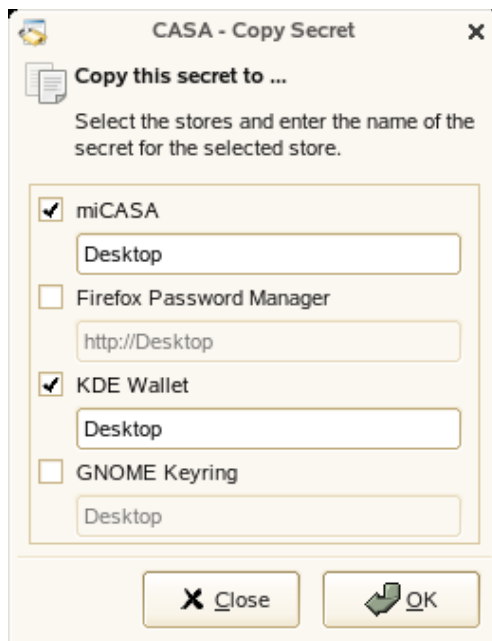
CASA enables you to copy existing user secrets from your miCASA store to other stores that are enabled in your system. Use the following procedure to copy secrets between credential stores that are enabled on the system.

- 1** In the Linux CASA Manager page, click the tab of any one of the stores that are enabled on your system. In this example, we've selected the miCASA tab.
- 2** Right-click the secret you want to copy in the Secret ID panel. In this example, we've selected the user's Desktop credentials as shown in the figure below:





- 3 In the *CASA - Copy Secret* window, check one or more of the available stores where you want to copy your secrets.





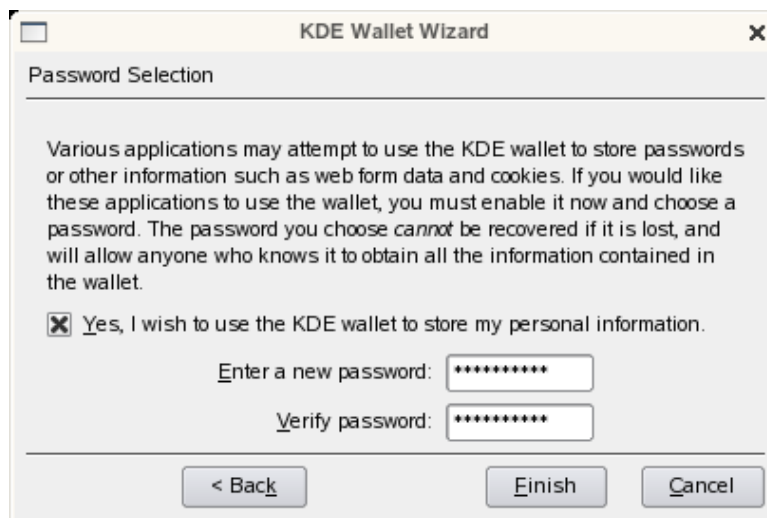
In this step, we've selected to copy secrets to the KDE Wallet store.

- 3a** Click *OK* and the *KDE Wallet Wizard* opens if the Wallet has not been configured previously.



KWallet enables you to store user secrets on a disk in an encrypted file. The wizard helps you to configure KWallet and we suggest you select the *Basic setup* option until you become familiar with the KDE Wallet system.

- 3b** Click the *Next* button to open the Password Selection window of the KDE Wallet Wizard.



If you want your applications to use KDE Wallet to store user secrets, you enable it by checking *Yes* in the Password Selector wizard and entering a *new password* and *password verification* in the fields.

---

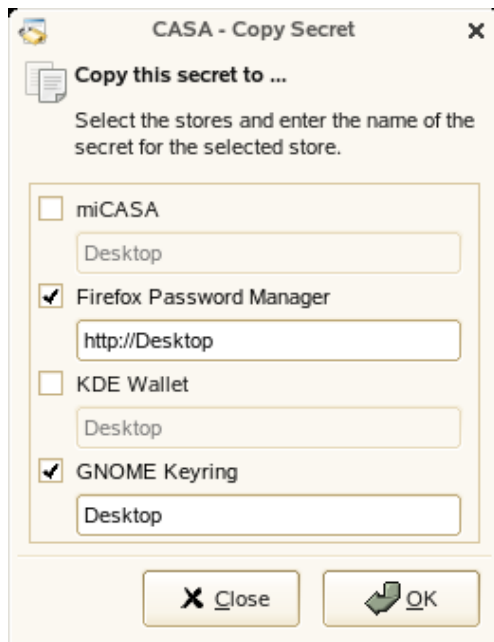
**NOTE:** To enhance security of your system, we recommend something other than your master password. The password you choose cannot be recovered if it is lost and will allow anyone who knows it to obtain all of the information contained in KDE Wallet.

---

- 3c** Click *Finish* and the KDE Daemon will open and prompt you for the password you created in [Step 5](#).
- 3d** Enter the password and click *OK*. An information window will indicate that the secrets were copied successfully.
- 3e** Click *Close*.

For copying secrets to credential stores other than KDE Wallet, use the following procedure:

- 1** Click the credential store into which you want to copy the secrets selected in [Step 2](#).



- 2** Change the name of the secret if needed.
- 3** Click *OK*. The Information advisory window will return a *Success* message.
- 4** Click *Close*.

### 4.3.10 Finding and Replacing Secrets

This feature enables you to find any password or value saved in the miCASA credential store and change it to any value you desire. To search all values, enter an asterisk (\*):

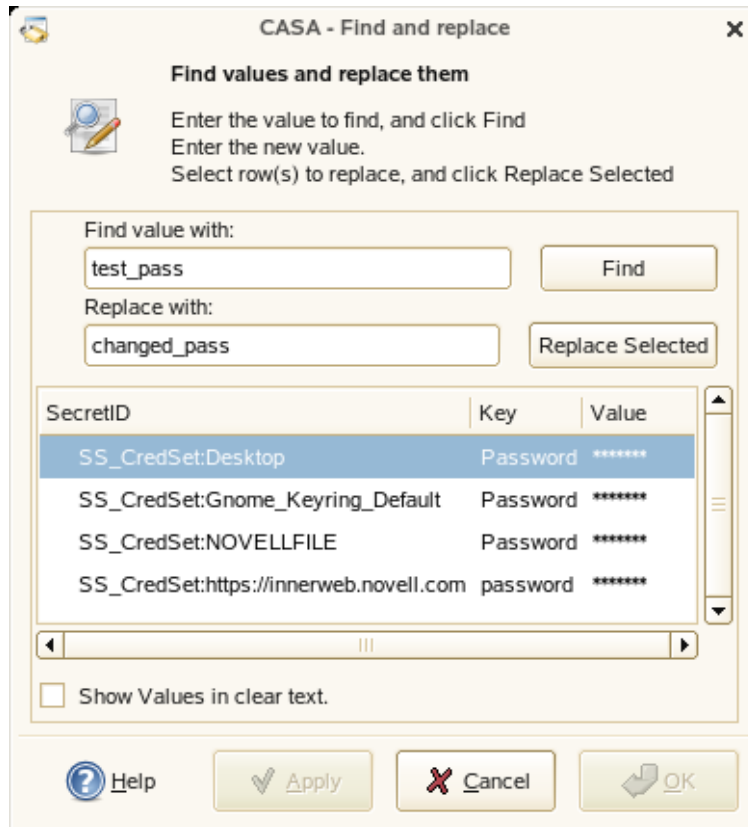
- 1** In CASA Manager, click *Edit > Find and Replace (Ctrl+F)*.
- 2** Enter your *master password* to confirm, then click *OK*. The *CASA Find and Replace* window opens:



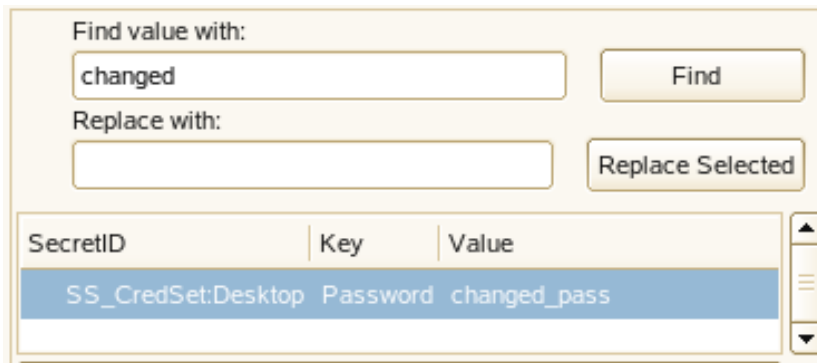
In the SecretID pane, the utility lists every credential and location where the value (*test\_pass*) is found. If you want to view the value in clear text, click the *Show Values in clear text* box.

- 4 To replace the existing password value, select one or more of the secrets listed in the SecretID panel.

You can select a single secret by clicking it or select multiple secrets by using Ctrl+click, Shift+click, or Ctrl+A.



- 5 Click the *Replace Selected* button, then click the *Apply* button to save the new value or click the *Revert* button to revert to the old value without saving the new value.
- 6 Click the *Cancel* or *OK* button when you are finished using the utility.
- 7 To verify the change, in this example, follow [Step 1](#) through [Step 3](#). In Step 3, type “*changed*” in the *Find value with:* after clicking the clear text box and entering your master password.



The new value you set (*changed\_pass*) has replaced your original password value.

---

**IMPORTANT:** If you are using the examples in this documentation to test the functionality of CASA Manager, make sure you revert any test password values back to your original values.

---

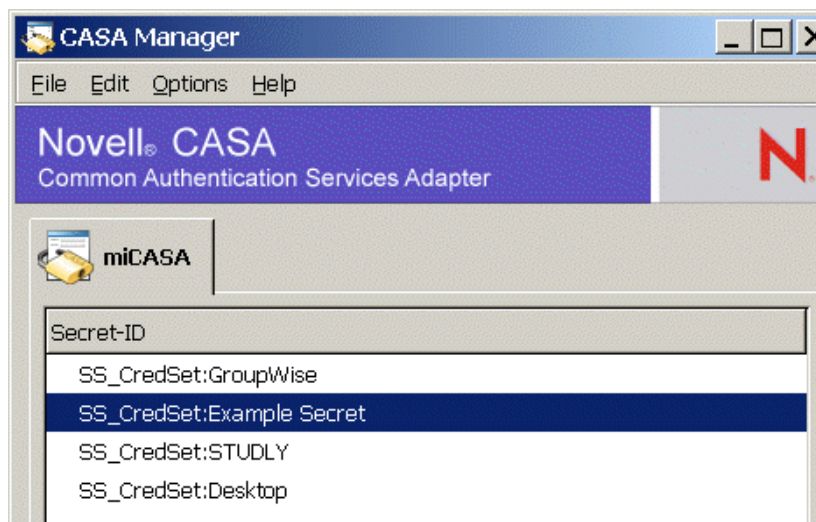
### 4.3.11 Editing Secrets

---

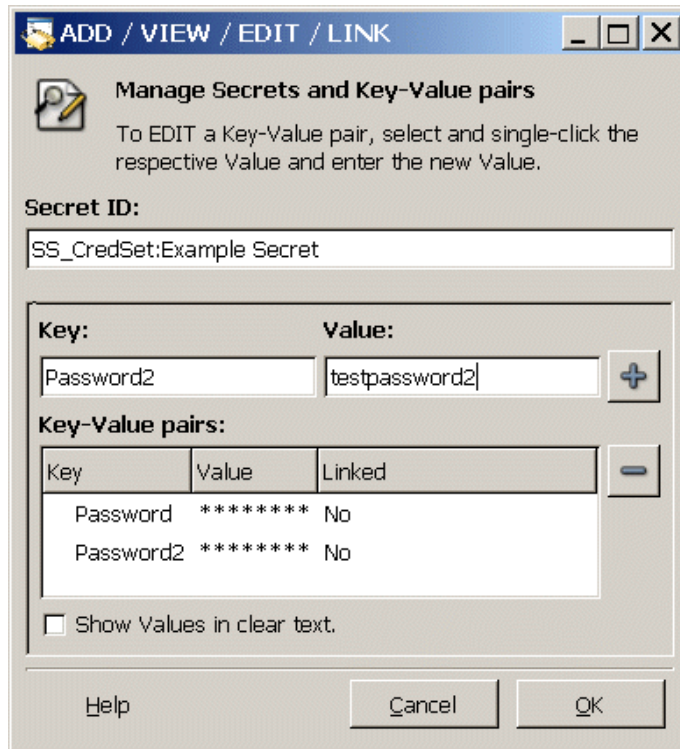
**NOTE:** The *Copy Secrets* feature is available in CASA 1.7 or greater.

---

- 1 Click the tab of the credential store you want to manage.
- 2 In the main Secret-ID window, double-click the secret you want to edit.



- 3 Edit the secret by adding new or changing existing key-value pairs.



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, click *Show Values in clear text*. You are then prompted to enter your master password before the values are displayed.

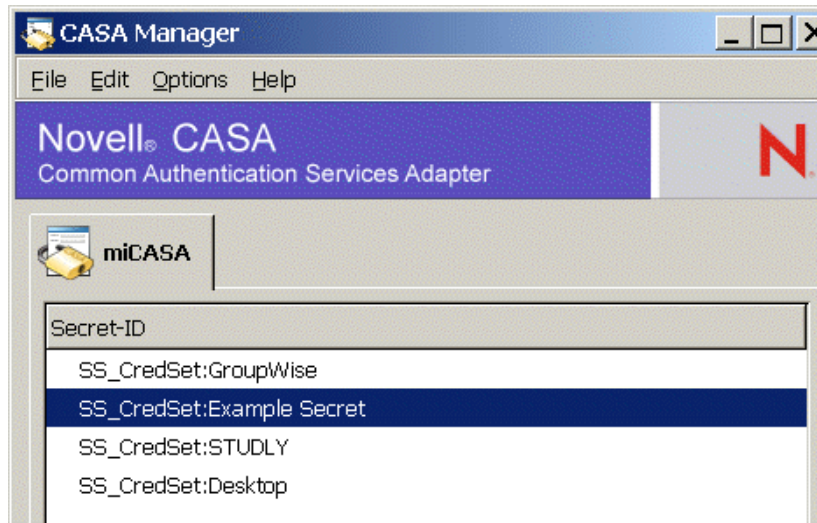
- 4 To edit the password value, click the *Value* field, type your new value, then click *OK*.

The new password value is saved in the miCASA credential store.

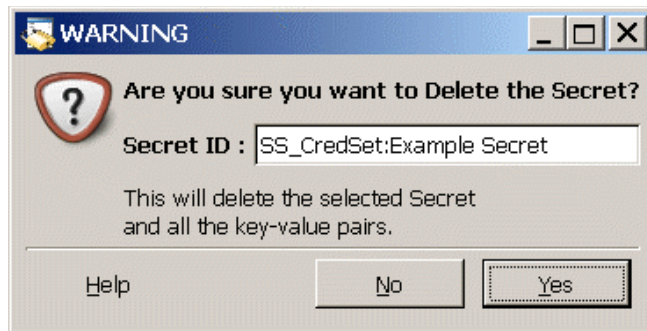
After they are created, Secret ID names cannot be edited.

### 4.3.12 Deleting Secrets

- 1 Click the tab of the credential story you want to manage.
- 2 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.



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



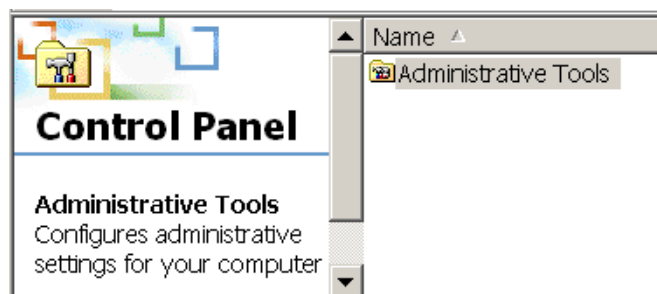
## 4.4 Resetting the CASA Master Password

There are instances when you might need to reset a user's master password, such as when they forget it. You do this by deleting all of the user's stored miCASA credentials. After performing the following procedure, CASA Manager will request the user to re-enter a new master password described in [Section 3.2.3, "Accessing CASA Manager,"](#) on page 20:

- 1 Shut down the CASA service.

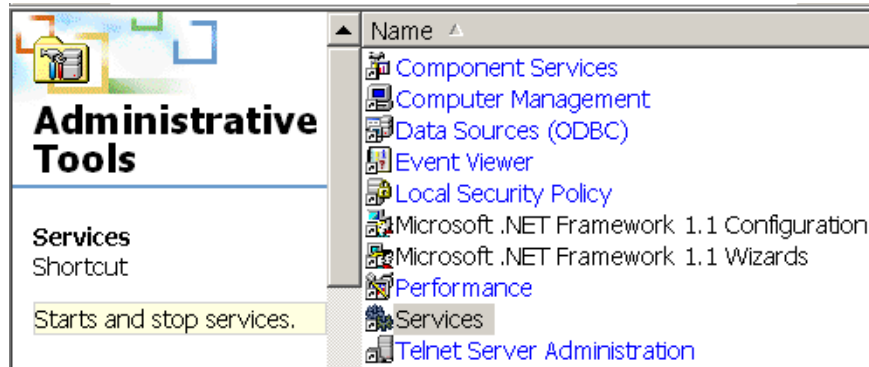
*In Windows:*

1. In the Windows 2000 control bar, click *Start > Settings > Control Panel*.

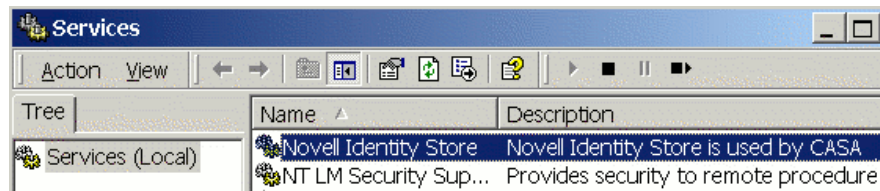


In Windows XP, click *Start > Control Panel > Administration Tools > Services*.

2. In Windows 2000, double-click *Administrative Tools*, then double-click *Services*.



3. Stop the Novell Identity Store service either by:  
Right-clicking on the service and selecting *Stop*  
Clicking on the service and then clicking the *Action* menu > *Stop*  
or  
Clicking the *Stop Service* button icon in the Services tool bar ■



Windows will stop the CASA identity store, which is necessary before you can remove the user's credentials explained in [Step 2](#). Leave the Services panel open because you will restart the Novell Identity Store after you delete the user's old credentials in [Step 2](#).

*In Linux:*

1. Stop the micasad service by running the following command as the root user:  

```
/etc/init.d/micasad stop
```
2. Locate the user's Home directory where the miCASA data files are stored.

---

**IMPORTANT:** Be certain your system settings are configured to Show hidden files and folders or the .miCASA\* files will not be viewable.

---

*In Windows 2000:*

1. Right-click on the *My Computer* desktop icon > *Open*.
2. In the My Computer address field, type *C:\Documents and Settings\[user's name]\casa*.  
This will list all of the documents and settings associated with the home user.







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

- ♦ “miCASAGetCredential” on page 51
- ♦ “miCASAOpenSecretStoreCache” on page 52
- ♦ “miCASAResReadBinaryKey” on page 53
- ♦ “miCASAResReadKey” on page 54
- ♦ “miCASARemoveCredential” on page 55
- ♦ “miCASARemoveKey” on page 56
- ♦ “miCASASetCredential” on page 57
- ♦ “miCASAWriteBinaryKey” on page 58
- ♦ “miCASAWriteKey” on page 60

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

For a list of possible error codes, see the `micasa_mgmd.h` header file located in the default install directory or [Appendix A, “CASA Error Codes,” on page 65](#).

## 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	The basic structure consists of a Username and a Password (see the <code>micasa_mgmd.h</code> header file).
SSCS_CRED_TYPE_SERVER_F	This credential type is used by services running on the workstation. Credentials stored and retrieved using this flag are not visible by CASA Manager, and are available after a reboot without user authentication.  On Linux install the CASA command line interface (CLI) package to manage these credentials.
SSCS_CRED_TYPE_BINARY_F	The binary structure consists of a Key ID, the binary data, and a length parameter (see the <code>micasa_mgmd.h</code> header file).

## credential

(OUT) Points to the credential structure [SSCS\\_BASIC\\_CREDENTIAL](#) (page 63).

## 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 no other credential is available.

## miCASAOpenSecretStoreCache

Allows an application to open a SecretStore cache.

## Syntax

```
SSCS_GLOBAL_LIBCALL_PTR(void) miCASAOpenSecretStoreCache
(
    SSCS_SECRETSTORE_T    *ssid,
    uint32_t              ssFlags,
    SSCS_EXT_T             *ext
);
```

## Parameters

### **ssid**

(IN) Points to the SecretStore structure found in `micasa.h` that defines the name of the store to be opened. For this release, use the default value defined in the `micasa.h` header file.

### **ssFlags**

(IN) Set to 0 for this release.

### **ext**

Reserved for future use.

## miCASARReadBinaryKey

Allows an application to read a key for a secret that contains binary data.

## Syntax

```
int32_t miCASARReadBinaryKey
(
    void                *context,
    uint32_t             ssFlags,
    SSCS_KEYCHAIN_ID_T   *keyChainID,
    SSCS_SECRET_ID_T     *sharedSecretID,
    SS_UTF8_T            *key,
    uint32_t             keyLen,
    uint8_t              *val,
    uint32_t             valLen,
    SSCS_PASSWORD_T      *epPassword,
    uint32_t             *bytesRequired,
    SSCS_EXT_T           *ext
);
```

## Parameters

### **context**

Points to the SecretStore cache by calling `miCASAOpenSecretStoreCache`.

### **ssFlags**

(IN) Set to 0 for this release.

### **keyChainID**

(IN) Points to the KeyChain ID structure defined by `SSCS_KEYCHAIN_ID_T`. For this release, set to the `SS_UTF8_T` `SSCS_SESSION_KEY_CHAIN_ID` defined in the header file `micasa.h`.

### **sharedSecretID**

(IN) Points to a structure of the secret ID for which the key is stored. This ID allows multiple applications to find and store a shared credential, such as Novell\_Collaboration.

**key**

(IN) Points to the name of the key to read. This key name must be in UTF-8 format and must be NULL terminated.

**keyLen**

(IN) Specifies the length of the key defined above, including the NULL terminator.

**val**

(IN/OUT) Char pointer to a buffer for which the value of a key is copied.

**valLen**

(IN/OUT) Pointer to the length of the buffer passed to receive the value for the key.

**epPassword**

Reserved for future use.

**bytesRequired**

Specifies the size of the buffer required to receive the value. If the buffer passed in is too small, an error is returned (see [Appendix A, “CASA Error Codes,” on page 65](#)).

**ext**

Reserved for future use.

## Return Values

If successful, returns the value of the key requested. If the key is not found or the buffer is too small, returns an appropriate error. For error code information, see the `micasa_mgmd.h` header file or see [Appendix A, “CASA Error Codes,” on page 65](#).

## miCASARReadKey

Allows an application to read a key or a secret containing character data.

### Syntax

```
int32_t miCASARReadKey
(
    void                *context
    uint32_t            ssFlags,
    SCS_KEYCHAIN_ID_T   *keyChainID,
    SCS_SECRET_ID_T     *sharedSecretID,
    SS_UTF8_T           *key,
    uint32_t            keyLen,
    uint8_t             *val,
    uint32_t            *valLen,
    SCS_PASSWORD_T      *epPassword,
    uint32_t            *bytesRequired,
    SCS_EXT_T           *ext
);
```

## Parameters

### **context**

Points to the SecretStore cache by calling [miCASAOpenSecretStoreCache](#).

### **ssFlags**

(IN) Set to 0 for this release.

### **keyChainID**

(IN) Points to the KeyChain ID structure defined by `SSCS_KEYCHAIN_ID_T`. For this release, set to `SS_UTF8_T SSCS_SESSION_KEY_CHAIN_ID` defined in the header file `micasa.h`.

### **sharedSecretID**

(IN) Points to a structure of the secret ID for which the key is stored. This ID allows multiple applications to find and store a shared credential, such as `Novell_Collaboration`.

### **key**

(IN) Points to the name of the key to read. This key name must be in UTF-8 format and NULL terminated.

### **keyLen**

(IN) Specifies the length of the key defined above, including the NULL terminator.

### **val**

(IN/OUT) Char pointer to a buffer for which the value of a key is copied.

### **valLen**

(IN/OUT) Pointer to the length of the buffer passed to receive the value for the key.

### **epPassword**

Reserved for future use.

### **bytesRequired**

Specifies the size of the buffer required to receive the value. If the buffer passed in is too small, an error is returned (see [Appendix A, “CASA Error Codes,” on page 65](#)).

### **ext**

Reserved for future use.

## Return Values

If successful, returns the value of the key requested. If the key is not found or the buffer is too small, returns an appropriate error. For error code information, see the `micasa_mgmd.h` header file or see [Appendix A, “CASA Error Codes,” on page 65](#).

## 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 (see [Appendix A, “CASA Error Codes,”](#) on page 65).

## miCASARemoveKey

Allows an application to remove a key from a secret stored in the cache.

## Syntax

```
int32_t miCASARemoveKey
(
    void          *context,
    uint32_t      ssFlags,
    SSCS_KEYCHAIN_ID_T *keyChainID,
    SSCS_SECRET_ID_T *sharedSecretID,
    SS_UTF8_T      *key,
    uint32_t      keyLen,
    SSCS_PASSWORD_T *epPassword,
    SSCS_EXT_T      *ext
);
```



## Parameters

### context

Points to the SecretStore cache by calling [miCASAOpenSecretStoreCache](#) (page 52).

### ssFlags

(IN) Set to 0 for this release.

### keyChainID

(IN) Points to the KeyChain ID structure defined by `SSCS_KEYCHAIN_ID_T`. For this release, set to `SS_UTF8_T SSCS_SESSION_KEY_CHAIN_ID` defined in the header file `micasa.h`.

### sharedSecretID

(IN) Points to a structure of the Secret-ID or name.

### key

(IN) Points to the name of the key to read. This key name must be in UTF-8 format and must be NULL terminated.

### keyLen

(IN) Specifies the length of the key defined above, including the NULL terminator.

### epPassword

Reserved for future use.

### ext

Reserved for future use.

## Return Values

If successful, set a credential and returns 0, or returns an error code (see [Appendix A, “CASA Error Codes,”](#) on page 65).

## 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, or returns an error code (see [Appendix A, “CASA Error Codes,”](#) on page 65).

## 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.

## miCASAWriteBinaryKey

Allows an application to write a binary key.

## Syntax

```
int32_t miCASAWriteBinaryKey
(
    void                *context,
    uint32_t            ssFlags,
    SSCS_KEYCHAIN_ID_T  *keyChainID,
    SSCS_SECRET_ID_T    *sharedSecretID,
    SS_UTF8_T           *key,
    uint32_t            keyLen,
    uint8_t             *val,
    uint32_t            valLen,
    SSCS_PASSWORD_T     *epPassword,
    SSCS_EXT_T          *ext
);
```

## Parameters

### context

Points to the SecretStore cache by calling [miCASAOpenSecretStoreCache \(page 52\)](#).

### ssFlags

(IN) Set to 0 for this release.

### keyChainID

(IN) Points to the KeyChain ID structure defined by SSCS\_KEYCHAIN\_ID\_T. For this release, set to SS\_UTF8\_T SSCS\_SESSION\_KEY\_CHAIN\_ID defined in the header file micasa.h.

### sharedSecretID

(IN) Points to a structure containing the Secret ID.

### key

(IN) Points to the buffer containing the key name, which must be in UTF-8 format and must be NULL terminated.

### keyLen

The length of the key, including the NULL terminator.

### val

Points to the buffer containing the binary data being saved in the cache.

### valLen

Points to the length of the buffer.

### epPassword

Reserved.

### ext

Reserved.

## Return Values

Sets the value for the given key and returns 0 if successful, or returns an error code (see [Appendix A, “CASA Error Codes,” on page 65](#)).

## miCASAWriteKey

Allows an application to write a key.

### Syntax

```
int32_t miCASAWriteKey
(
    void                *context,
    uint32_t            ssFlags,
    SSCS_KEYCHAIN_ID_T  *keyChainID,
    SSCS_SECRET_ID_T    *sharedSecretID,
    SS_UTF8_T           *key,
    uint32_t            keyLen,
    uint8_t             *val,
    uint32_t            valLen,
    SSCS_PASSWORD_T     *epPassword,
    SSCS_EXT_T          *ext
);
```

### Parameters

#### context

Points to the SecretStore cache by calling miCASAOOpenSecretStoreCache.

#### ssFlags

(IN) Set to 0 for this release.

#### keyChainID

(IN) Points to the KeyChain ID structure defined by SSCS\_KEYCHAIN\_ID\_T. For this release, set to SS\_UTF8\_T SSCS\_SESSION\_KEY\_CHAIN\_ID defined in the header file micasa.h.

#### sharedSecretID

(IN) Points to a structure containing the Secret ID.

#### key

(IN) Points to the buffer containing the Key name, which must be in UTF-8 format and must be NULL terminated.

#### keyLen

The length of the key, including the NULL terminator.

#### val

Points to the buffer containing the binary data being saved in the cache.

**valLen**

Points to the length of the buffer.

**epPassword**

Reserved.

**ext**

Reserved.

## **Return Values**

Sets the value for the given key and returns 0 if successful, or returns an error code (see [Appendix A, “CASA Error Codes,” on page 65](#)).



# Structures

# 6

CASA uses the following structures:

- ♦ “SSCS\_BASIC\_CREDENTIAL” on page 63
- ♦ “SSCS\_SECRET\_ID\_T” on page 63

## 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.



# CASA Error Codes

# A

Dec Value	Hexadecimal Value	Name	Error Text
0	0x00000000	NSSCS_SUCCESS	The requested function completed successfully.
-800	0xFFFFFCE0	NSSCS_E_OBJECT_NOT_FOUND	Can't find the target object DN in eDirectory. (Resolve name failed.)
-801	0xFFFFCFDF	NSSCS_E_NICI_FAILURE	The NICI encryption operations have failed.
-802	0xFFFFCFDE	NSSCS_E_INVALID_SECRET_ID	The secret ID is not in the user SecretStore.
-803	0xFFFFCFDD	NSSCS_E_SYSTEM_FAILURE	Some internal operating system services have not been available.
-804	0xFFFFCFDC	NSSCS_E_ACCESS_DENIED	Access to the target SecretStore has been denied.
-805	0xFFFFCFDB	NSSCS_E_NDS_INTERNAL_FAILURE	Some internal eDirectory services are not available.
-806	0xFFFFCFDA	NSSCS_E_SECRET_UNINITIALIZED	A secret has not been initialized with a write.
-807	0xFFFFCFD9	NSSCS_E_BUFFER_LEN	The size of the buffer is not in a nominal range between minimum and maximum values.
-808	0xFFFFCFD7	NSSCS_E_CORRUPTED_STORE	Versions of the client and server components are not compatible.
-809	0xFFFFCFD7	NSSCS_E_CORRUPTED_STORE	SecretStore data on the server has been corrupted.
-810	0xFFFFCFD6	NSSCS_E_SECRET_ID_EXISTS	The secret ID already exists in the SecretStore.
-811	0xFFFFCFD5	NSSCS_E_NDS_PWORD_CHANGED	The user's eDirectory password has been changed by the administrator.
-812	0xFFFFCFD4	NSSCS_E_INVALID_TARGET_OBJECT	The target eDirectory user object is not found.
-813	0xFFFFCFD3	NSSCS_E_STORE_NOT_FOUND	The target eDirectory user object does not have a SecretStore.
-814	0xFFFFCFD2	NSSCS_E_SERVICE_NOT_FOUND	The SecretStore is not on the Network.
-815	0xFFFFCFD1	NSSCS_E_SECRET_ID_TOO_LONG	The length of the secret ID buffer exceeds the limit.

Dec Value	Hexadecimal Value	Name	Error Text
-816	0xFFFFFCD0	NSSCS_E_ENUM_BUFF_TOO_SHORT	The length of the enumeration buffer too short.
-817	0xFFFFFCCF	NSSCS_E_NOT_AUTHENTICATED	The user is not authenticated.
-818	0xFFFFFCCE	NSSCS_E_NOT_SUPPORTED	The operation is not supported.
-819	0xFFFFFCCD	NSSCS_E_NDS_PWORD_INVALID	The eDirectory password entered is not valid.
-820	0xFFFFFCCC	NSSCS_E_NICI_OUTOF_SYNC	The session keys of the client and server NICI are out of sync.
-821	0xFFFFFCCB	NSSCS_E_SERVICE_NOT_SUPPORTED	The requested service is not yet supported.
-822	0xFFFFFCCA	NSSCS_E_TOKEN_NOT_SUPPORTED	The eDirectory authentication type is not supported.
-823	0xFFFFFCC9	NSSCS_E_UNICODE_OP_FAILURE	The Unicode text conversion operation failed.
-824	0xFFFFFCC8	NSSCS_E_TRANSPORT_FAILURE	The server connection is lost.
-825	0xFFFFFCC7	NSSCS_E_CRYPTOP_OP_FAILURE	The cryptographic operation failed.
-826	0xFFFFFCC6	NSSCS_E_SERVER_CONN_FAILURE	Opening a connection to the server failed.
-827	0xFFFFFCC5	NSSCS_E_CONN_ACCESS_FAILURE	Access to a server connection failed.
-828	0xFFFFFCC4	NSSCS_E_ENUM_BUFF_TOO_LONG	The size of the enumeration buffer exceeds the limit.
-829	0xFFFFFCC3	NSSCS_E_SECRET_BUFF_TOO_LONG	The size of the secret buffer exceeds the limit.
-830	0xFFFFFCC2	NSSCS_E_SECRET_ID_TOO_SHORT	The length of the Secret ID should be greater than zero.
-831	0xFFFFFCC1	NSSCS_E_CORRUPTED_PACKET_DATA	The protocol data was corrupted on the wire.
-832	0xFFFFFCC0	NSSCS_E_EP_ACCESS_DENIED	The EP password validation failed, so access to the secret was denied.
-833	0xFFFFFCBF	NSSCS_E_SCHEMA_NOT_EXTENDED	The schema is not extended to support SecretStore on the target tree.
-834	0xFFFFFCBE	NSSCS_E_ATTR_NOT_FOUND	One of the optional service attributes is not instantiated.
-835	0xFFFFFCBD	NSSCS_E_MIGRATION_NEEDED	The server has been upgraded, so the user SecretStore should be updated.

Dec Value	Hexadecimal Value	Name	Error Text
-836	0xFFFFFCBC	NSSCS_E_MP_PWORD_INVALID	The master password could not be verified to read or unlock the secrets.
-837	0xFFFFFCBB	NSSCS_E_MP_PWORD_NOT_SET	The master password has not been set on the SecretStore.
-838	0xFFFFFCBA	NSSCS_E_MP_PWORD_NOT_ALLOWED	The ability to use a master password has been disabled.
-839		NSSCS_E_WRONG_REPLICA_TYPE	There's no writable replica of eDirectory.
-840	0xFFFFFCB9	NSSCS_E_ATTR_VAL_NOT_FOUND 0xFFFFFCB8	The target attribute is not instantiated in eDirectory.
-841	0xFFFFFCB7	NSSCS_E_INVALID_PARAM	The API parameter is not initialized.
-842	0xFFFFFCB6	NSSCS_E_NEED_SECURE_CHANNEL	The connection to the SecretStore needs to be over SSL.
-843	0xFFFFFCB5	NSSCS_E_CONFIG_NOT_SUPPORTED	No server to support the given override configuration is found.
-844	0xFFFFFCB4	NSSCS_E_STORE_NOT_LOCKED	The attempt to unlock SecretStore failed because the store is not locked.
-845	0xFFFFFCB3	NSSCS_E_TIME_OUT_OF_SYNC	The eDirectory replica on the server that holds SecretStore is out of sync with the replica ring.
-846	0xFFFFFCB2	NSSCS_E_VERSION_MISMATCH	The versions of the client DLLs don't match.
-847	0xFFFFFCB1	NSSCS_E_SECRET_BUFF_TOO_SHORT	The buffer supplied for the secret is too short (minimum NSSCS_MIN_IDLIST_BUF_LEN).
-848	0xFFFFFCB0	NSSCS_E_SH_SECRET_FAILURE	The shared secret's processing and operations failed.
-849	0xFFFFFCAF	NSSCS_E_PARSER_FAILURE	The shared secret's parser operations failed.
-850	0xFFFFFCAE	NSSCS_E_UTF8_OP_FAILURE	The Utf8 string operations failed.
-851	0xFFFFFCAD	NSSCS_E_CTX_LESS_CN_NOT_UNIQUE	The contextless name for LDAP bind does not resolve to a unique DN.
-852	0xFFFFFCAC	NSSCS_E_UNSUPPORTED_BIND_CRED	The login credential for advanced bind is not supported.
-853	0xFFFFFCAB	NSSCS_E_CERTIFICATE_NOT_FOUND	The LDAP root certificate required for bind operations was not found.
-855	0xFFFFFCA9	NSSCS_E_WRONG_SH_SEC_TYPE	The shared secret tag is unrecognized or unknown.

Dec Value	Hexadecimal Value	Name	Error Text
-888	0xFFFFFC88	NSSCS_E_NOT_IMPLEMENTED	The feature is not implemented yet.
-899	0xFFFFFC7D	NSSCS_E_BETA_EXPIRED	The product's beta life has expired. Purchase an official release copy.

# CASA Security Guidelines

# B

All CASA users should review the Security Best Practices Guide before reading this document.

This section contains an overview of the security guidelines that developers, administrators, and users should consider when developing, deploying, and using CASA:

- ♦ [Section B.1, “CASA Security Administration,” on page 69](#)
- ♦ [Section B.2, “CASA User Security,” on page 70](#)
- ♦ [Section B.3, “Security Considerations for CASA Developers,” on page 71](#)

## B.1 CASA Security Administration

The CASA Credential Store (miCASA), a subcomponent of the CASA application appended to the operating system, securely stores user credentials.

CASA is designed to decrypt and load persistent secrets into memory only after the user has supplied the desktop password or the master password to authenticate and start the service. At least one of these passwords is required to decrypt the persistent files before the secrets are cached in memory for the logged-in user. Once set, the master password is used to recover persistent secrets stored by CASA when the desktop password has changed or is forgotten by the user.

Because all user credential management is performed through programming interfaces by the CASAManager utility, the miCASA service typically does not require any specific administrator input.

However, if a user’s credentials become corrupted or, in a worst-case scenario, a user forgets both the desktop and master passwords the system administrator can use the following procedure to reset CASA:

- 1 Locate the user's hidden `.miCASA*` credential files.

These files are located either in the user’s home directory or, if the user's home directory is encrypted, CASA by default uses `/home/.casa/userid`. If the user has chosen to store these files on a mobile device, such as a USB drive, the administrator must contact the user to have those files deleted from that device.

- 2 Delete all `.miCASA*` files.

This causes the CASA service to restart with new files, which prompts the user to set a new master password.

---

**IMPORTANT:** All existing user secrets will be lost and recovery from backup is not possible.

---

- 3 Users must create a new master password.

For more detailed information, see [Section 4.4, “Resetting the CASA Master Password,” on page 47](#). For more information about setting up or managing CASA, [Chapter 4, “CASA Manager Administration,” on page 23](#).

## B.2 CASA User Security

This section provides some of the practices that should be considered to increase user security when deploying and using CASA. All users should read the Security Best Practices Guide before using CASA:

- ♦ **Store credentials securely:** If the security policies of your user environment allow, consider changing the default location of the CASA persistent files (where user's secrets are stored) to a removable device, such as a USB drive. This is done using CASAManager (CASAManager.exe.) Then, to ensure that the CASA service finds the files it needs, plug in the USB drive before logging in. This practice ensures that persistent files and user secrets are not left on the workstation.
- ♦ **Always use secure passwords:** Password should always contain eight or more characters, including uppercase and lowercase letters, numbers, special characters, etc.) For more information, refer to the Security Best Practices Guide.
- ♦ **Link secrets carefully:** Although CASA allows you to conveniently link secrets, this is not necessarily a best security practice.

Linking secrets essentially enables you to use the same password for multiple services. However, if one application is compromised, then all of the applications using that password are compromised and the security of every application sharing a linked password is reduced to the lowest common denominator that is the weakest security in the application chain.

This means an attacker needs to target only the application with the weakest security to break it and obtain access to all of the other applications using the same credential. After the attacker succeeds accessing one application, all of the applications in the chain are compromised.

- ♦ **Lock the CASA store when an enabled computer is unattended:** For more information, refer to the Security Best Practices Guide.
- ♦ **Log out of the desktop session when the session is over or when leaving the machine for an extended time.** For more information, refer to Security Best Practices Guide.
- ♦ **Back up secrets often and encrypt them in the process:** Always secure copies of the persistent files in a safe place.
- ♦ **Enhance Firefox browser protection:** In Firefox, set the master password for additional protection when using the Firefox tab in CASAManager to manage secrets in the Firefox store.
- ♦ **Disable *Remember Passwords* in Firefox** To prevent storing secrets in multiple locations when using the Firefox plug-in for CASA, disable the Firefox *Remember Passwords* feature and clear the current passwords stored in the Firefox cache. For security, passwords should be in one secure location.
- ♦ **Use Pluggable Authentication Modules (PAM) to capture login credentials on Linux:** Configuring CASA in YaST causes services such as ssh, gdm (Gnome Display Manager), kdm (KDE Display Manager), and login authentication to insert CASA into their authentication stacks. As a result, credentials are captured and stored in CASA when using these services.
- ♦ **Create different passwords for the Gnome Keyring and the Desktop:** For user convenience, the Gnome Keyring automatically stores its master password in CASA upon login to a Gnome session, and uses this password upon subsequent logins to unlock itself without requiring user interaction.

If the Gnome Keyring master password is not been set before CASA is enabled, the Gnome Keyring will use the desktop password as its master password. To enhance security, it is recommended that the user set the Gnome Keyring master password to something other than the desktop password before enabling CASA.

- ♦ **Do not install the CASA SDK until reading [Section B.3, “Security Considerations for CASA Developers,”](#) on page 71.**

## B.3 Security Considerations for CASA Developers

The CASA SDK should be installed only if necessary when developing and installing new CASA applications. In addition to demonstrating and testing new applications, the CASA SDK sample code binaries allow developers to review the secrets stored in the user’s persistent store without the necessary security safeguards required in a production environment (such as requiring users to enter the master password to view the values of the secrets in clear text within CASAManager).

Under development conditions, the master password is used only under certain conditions:

During authentication at the beginning of the session

or

When the CASAManager is accessed initially during the process of decrypting the data stored in the files and loading it into memory for the duration of the session.

After this stage, CASAManager controls access by requiring a pseudo authentication to allow access. However, the demonstration code in the SDK still functions without safeguards after the initial authentication.





# Revision History



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

---

April 05, 2011	<ul style="list-style-type: none"><li>♦ Updated <a href="#">Chapter 2, “CASA on Linux,”</a> on page 11.</li><li>♦ Updated <a href="#">Chapter 3, “CASA on Windows,”</a> on page 19.</li></ul>
September 28, 2007	<ul style="list-style-type: none"><li>♦ Refreshed documentation for delivery of Novell Open Enterprise Server 2.</li><li>♦ Added <a href="#">Section 2.2.6, “Avoiding Conflicting Service Realms,”</a> on page 16.</li><li>♦ Added <a href="#">Section 2.2.7, “Using CASA in chroot Environments,”</a> on page 17.</li></ul>
May 15, 2007	Updated CASA documentation and deliverables from version 1.6 to 1.7.  Added <a href="#">Appendix B, “CASA Security Guidelines,”</a> on page 69.  Update CASA Help files to coincide with version 1.7.
October 11, 2006	Documented the following miCASA functions: <ul style="list-style-type: none"><li>♦ <a href="#">miCASAOpenSecretStoreCache</a> (page 52)</li><li>♦ <a href="#">miCASAReadBinaryKey</a> (page 53)</li><li>♦ <a href="#">miCASAReadKey</a> (page 54)</li><li>♦ <a href="#">miCASARemoveKey</a> (page 56)</li><li>♦ <a href="#">miCASAWriteBinaryKey</a> (page 58)</li><li>♦ <a href="#">miCASAWriteKey</a> (page 60)</li></ul>
May 22, 2006	Updated CASA documentation and deliverables from version 1.5 to 1.6.
November 18, 2005	<ul style="list-style-type: none"><li>♦ Updated CASA documentation and deliverables from version 1.0 to 1.5.</li><li>♦ Documented new CASA Manager functionality in <a href="#">Chapter 4, “CASA Manager Administration,”</a> on page 23.</li></ul>
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.

---

