Implementing Security Features Using Keychain

For Keychain 2.0

Preliminary

5/17/00

Contents

Figures, Listings, and Tables

C H A P T E R 1

Introduction

This document introduces the Keychain Manager. The Keychain Manager provides a uniform way to handle passwords for multiple users, multiple databases, or any situation in which a user must enter single or multiple passwords. You can use the Keychain Manager to provide secure storage for a user's passwords, cryptographic keys, and digital certificates. Your application should use Keychain Manager if it requires the user to enter a password for access to some document or service.

This document describes the Keychain Manager in the following sections:

- "Keychain Manager Concepts" (page 11) provides a conceptual overview of the Keychain Manager.
- "Keychain Manager Tasks" (page 23) provides an introduction to programming the Keychain Manager.
- "Keychain Manager Reference" (page 35) provides a complete description of the Keychain Manager 2.0 API, including its functions, data types, constants, and result codes.
- "API and Document Revision History" (page 113) provides a history of changes to this document, as well as changes to the Keychain Manager API from version 1.0.1 to 2.0.

All code listings in this document are shown in C, except for listings that describe resources, which are shown in Rez-input format. Many listings are taken from the MenuScripter sample application, which is available through Apple's developer website at

<http://developer.apple.com/>

Although the sample code in this document has been compiled and tested to some degree, Apple Computer does not recommend that you directly incorporate this code into your application. For example, only limited error handling is shown—you should develop your own techniques for detecting and handling errors.

C H A P T E R 2

Keychain Manager Concepts

This chapter introduces the Keychain Manager and describes how the user can use it to store passwords, use multiple keychains, access the keychain, provide secure storage for a user's passwords, cryptographic keys, and digital certificates.

The following sections provide a conceptual overview of Keychain Manager:

Keychain Manager Implementation

The first public release of Keychain Manager was version 2.0, which is supported on computers running Mac OS 9 or later. Keychain Manager 1.0.1 SDK was the first version of Keychain Manager released to developers. For a brief summary of the changes to the Keychain Manager API from 1.0.1 to 2.0, see "API and Document Revision History" (page 113).

Accessing the Keychain

A **keychain** is a secure repository for user-centric data, such as keys, passwords and certificates. The repository may be a file, a network database, a smart card, or other storage media. The Keychain Manager enables your application to store multiple passwords in a keychain. The keychain is accessed by one passphrase or master password. A **passphrase** is a master password that unlocks the keychain and allows applications to access all the user’s application and service passwords. The passphrase provides **transparent authentication** to the user, such that the user can access all their passwords in that keychain with a single sign on. The passphrase is not stored on the disk and is not accessible to applications.

Since a computer can be shared by more than one user, the Keychain Manager specifies a **default keychain**. The **default keychain** is the currently unlocked keychain to which new items are added. When the Keychain Manager is called and it detects that no default keychain is available, the user is prompted to create one via the Keychain Access application, described in “Interacting With the Keychain Access Application” (page 15). This application allows users to view and manage items in a keychain. After the user has created and unlocked a keychain, your application can begin using it.

The default keychain is automatically configured by the Keychain Manager when a keychain is created for the first time on the user's machine. If the default keychain isn't configured (that is, if there is no preference file for the default keychain in Internet Config), the user is prompted to choose among the locked keychains (first one chosen in the list). When an unspecified keychain is to be unlocked, the default keychain is automatically selected for the user. The user can modify the preference file of the default keychain using the Keychain Access application. For more information, see “Interacting With the Keychain Access Application” (page 15).

When the computer is started up, all keychains are locked. Keychains remain locked until the user establishes authentication and unlocks the keychain. Until this time, your application cannot access items in the keychain.

Keychain Items

A **keychain item** is a chunk of data with attached information that identifies attributes of a keychain item. The data and attributes in a keychain item are encrypted. Every active keychain has a lock interval timer, which locks the keychain automatically after a period of time that you specify. This feature is useful for security reasons. Without a lock interval timer, a user might forget to lock the keychain before leaving the computer, and anyone else with access to that computer can potentially access the user's passwords. When a keychain is unlocked, the user's passwords are available to any application running on their machine.

There are five different types of keychain items: AppleShare passwords, Internet passwords, generic passwords, cryptographic keys, and digital certificates. Each type of keychain item has a unique set of **attributes**, as well as some attributes that are common to all keychain items. Examples of common attributes include item type, creation date, description, modification date, and comments. The Keychain Manager has access to item attributes for the purposes of searching keychains. You can use keychain item attributes to perform such tasks as recording and tracking passwords the user has chosen in your application.

Digital certificates are stored in a user's keychain along with their own personal certificate and the root certificates they have allowed to be added. Digital certificates are used to refer to people the user knows.

Cryptographic keys can be symmetric or asymmetric. Symmetric keys are typically used for session or password-based encryption. This is synonymous with a shared secret key. These keys can only be generated using the Keychain Manager API.Asymmetric key pairs are used to do signing and encryption without a shared secret. When an asymmetric key pair is generated, it is typically associated with a digital certificate. Future versions of the Keychain Manager may provide the ability to create, store, and display symmetric asymmetric keys. These keys can currently be generated using the Certificate Assistant or Signing application.

Using Multiple Keychains

The Keychain Manager enables the user to open multiple keychains simultaneously. The user can use drag and drop to manipulate keychain contents. In addition, the user can drag certificates to and from the keychain, effectively importing or exporting them. This allows users to have their secure data spread across multiple keychains and be able to access the information no matter where it resides.

Searching Keychains

Prior to version 2.0 of this technology, the Keychain Manager only searched unlocked keychains. In version 2.0 and later, when trying to find a password, the Keychain Manager searches all unlocked keychains first, starting with the default keychain. If a match is found in more than one keychain, the user might be presented with a dialog asking them to select the item they want to use.

If a password is not found in an unlocked keychain, the Keychain Manager searches all locked keychains. If the password is found in a locked keychain, the user will be prompted to unlock the keychain so the password can be retrieved. If no match is found in any keychain, locked or unlocked, the Keychain Manager returns the result code `itemNotFound` and requires user interaction.

In Mac OS X, the user has the option of selecting the search order and the keychains to be searched.

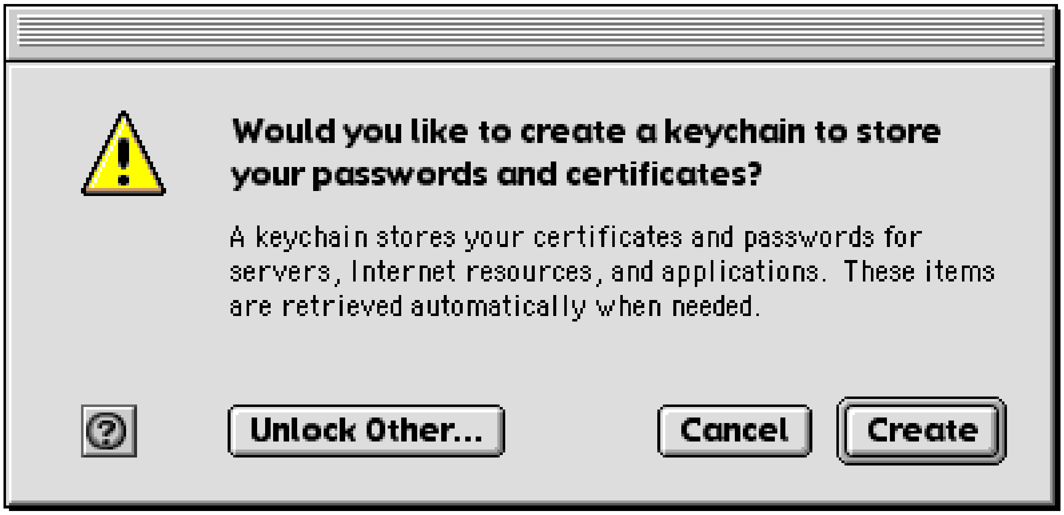
Interacting With the Keychain Access Application

Keychain Access is an application that enables the user to view and manage items in a keychain. Specifically, it allows the user to

- create a default keychain
- modify the preference file of the default keychain
- unlock a keychain
- view the items stored in a keychain

Figure 2-1 (page 15) shows the dialog that is displayed when the user first opens Keychain Access. Keychain Access prompts them to either create a keychain or unlock an existing one.

Figure 2-1 Create/Unlock Keychain dialog

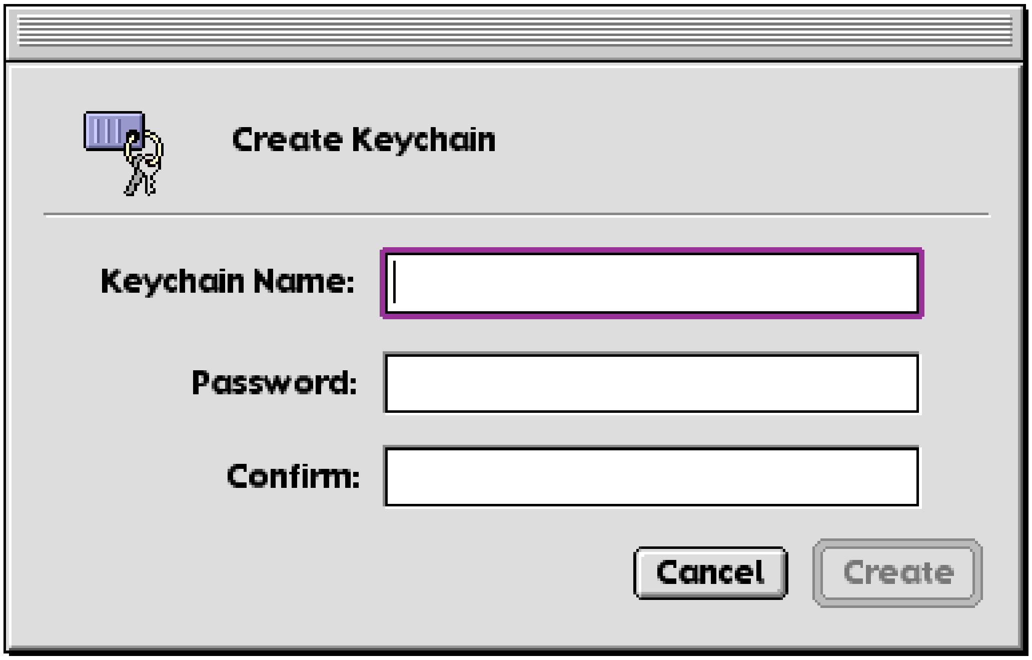


Would you like to create a keychain to store your passwords and certificates?

A keychain stores your certificates and passwords for servers, Internet resources, and applications. These items are retrieved automatically when needed.

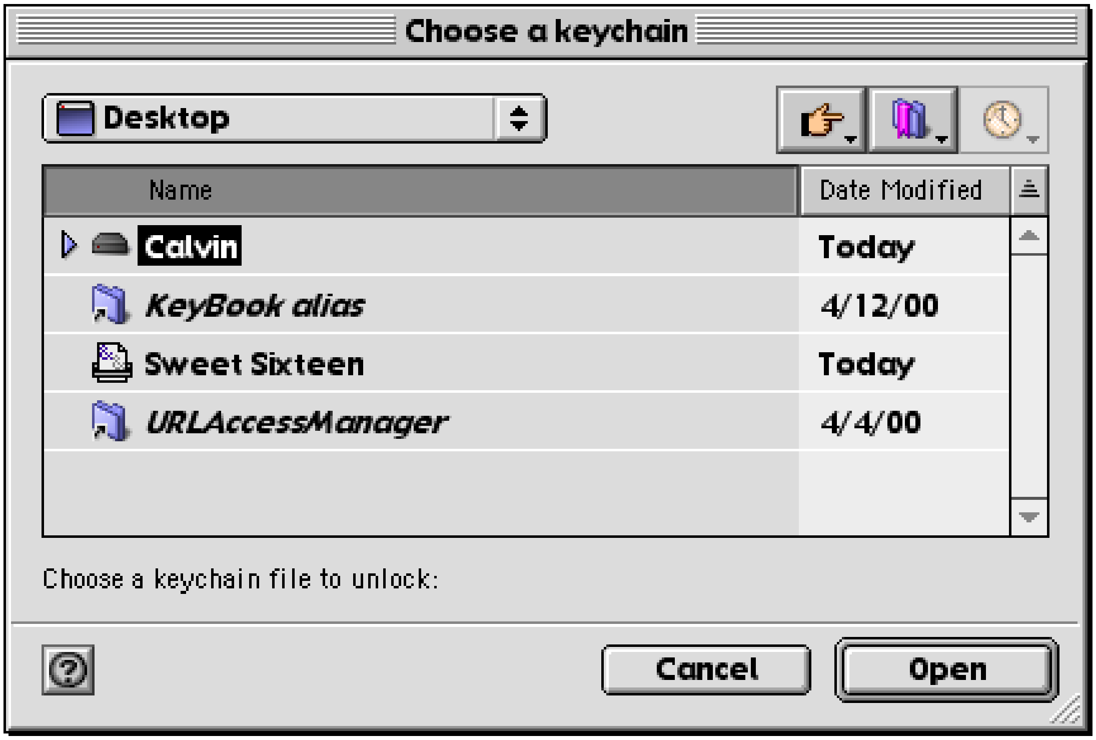
If the user has not yet created a keychain, they should select the Create button. In this case, Keychain Access displays the Create Keychain dialog shown in Figure 2-2 (page 16) prompting them to create a keychain.

Figure 2-2 Create Keychain dialog



If the user has already created a keychain, they should select the Unlock Other button. In this case, Keychain Access prompts them to select the keychain that they wish to unlock. Figure 2-3 (page 17) shows the Choose Keychain dialog.

Figure 2-3 Choose Keychain dialog

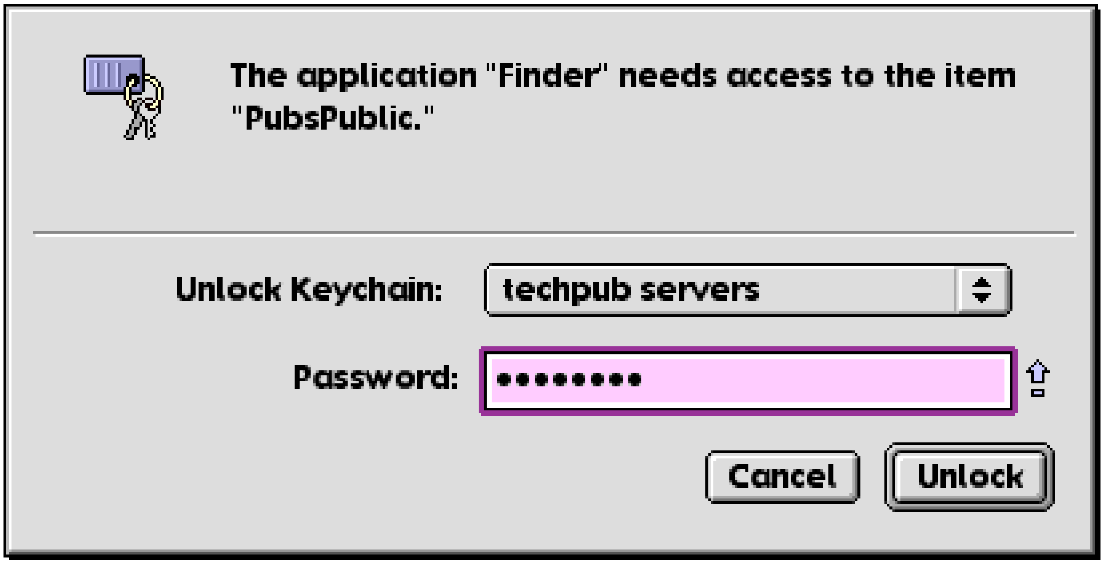


Name	Date Modified
Calvin	Today
KeyBook alias	4/12/00
Sweet Sixteen	Today
URLAccessManager	4/4/00

Choose a keychain file to unlock:

When the system or an application needs to access a password in the keychain, Keychain Access displays the Allow Access dialog shown in Figure 2-4 (page 18).

Figure 2-4 Allow Access dialog



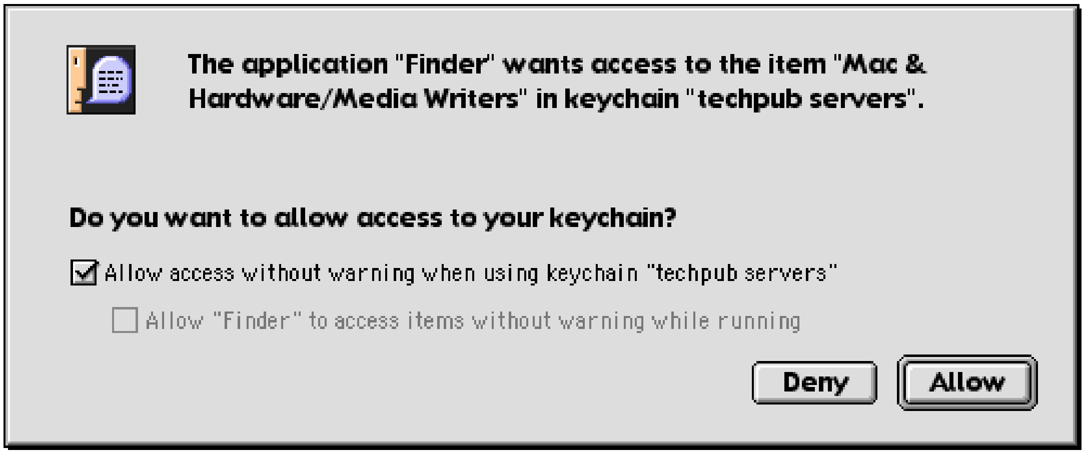
The application "Finder" needs access to the item "PubsPublic."

Unlock Keychain: techpub servers

Password: •••••••••

If the user selects the Unlock button, Keychain Access displays the Allow Unrestricted Access dialog shown in Figure 2-5 (page 19). This dialog prompts the user to indicate whether they wish to give an application unrestricted access to the contents of their keychain.

Figure 2-5 Allow Unrestricted Access dialog



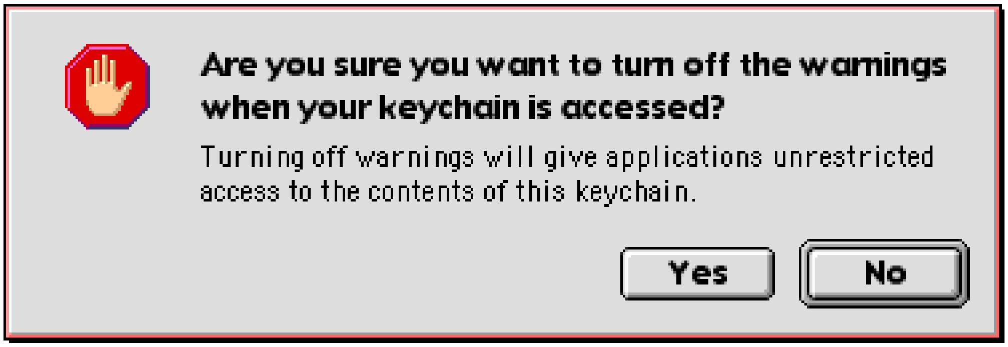
The application "Finder" wants access to the item "Mac & Hardware/Media Writers" in keychain "techpub servers".

Do you want to allow access to your keychain?

- [✓] Allow access without warning when using keychain "techpub servers"
- [ ] Allow "Finder" to access items without warning while running

If the user selects the Allow button, the Turn Off Warnings alert shown in Figure 2-6 (page 20) is displayed. This alert warns the user of the repercussoins of allowing access to the keychain without warning.

Figure 2-6 Turn Off Warnings alert

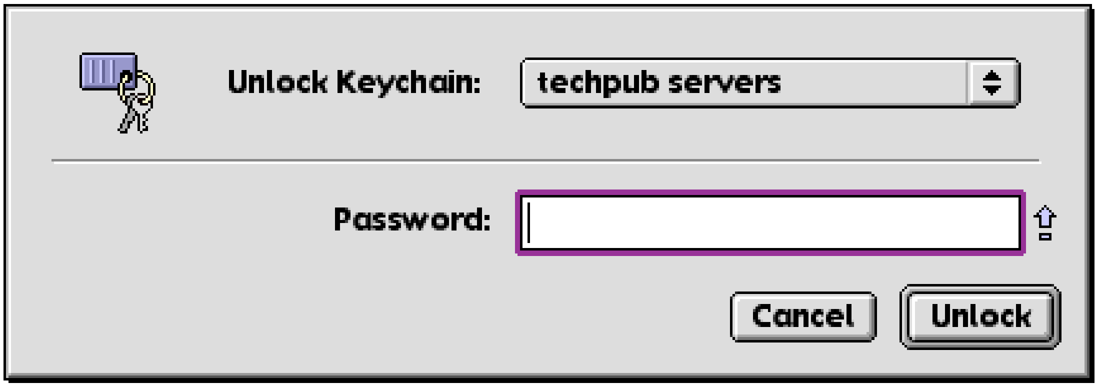


Are you sure you want to turn off the warnings when your keychain is accessed?

Turning off warnings will give applications unrestricted access to the contents of this keychain.

If a keychain already exists when the user first opens Keychain Access, the dialog shown in Figure 2-7 (page 20) is displayed. The Unlock Keychain dialog prompts the user for their password.

Figure 2-7 Unlock Keychain dialog

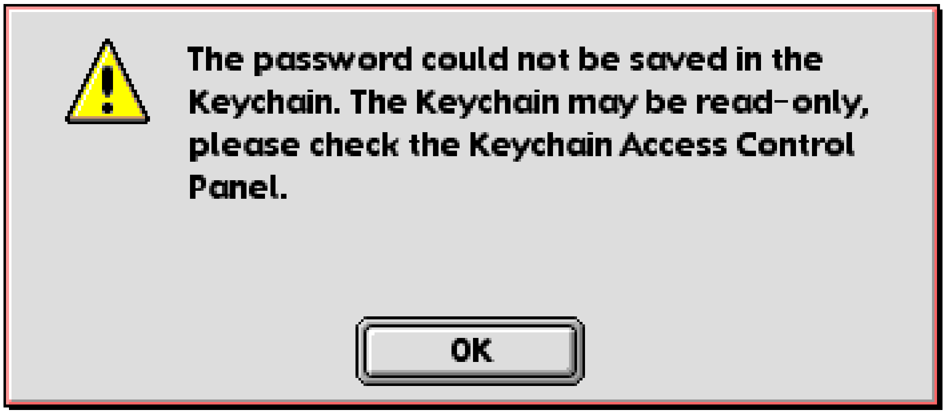


Unlock Keychain: techpub servers

Password:

If the user previously marked the keychain as read-only in the Keychain Access application, the Read-only Keychain alert shown in Figure 2-8 (page 21) is displayed. In this case, the password for the keychain cannot be saved.

Figure 2-8 Read-only Keychain alert



The password could not be saved in the Keychain. The Keychain may be read-only, please check the Keychain Access Control Panel.

Keychain Manager Tasks

This chapter describes how you can modify your application to handle passwords, cryptographic keys, and digital certificates using the Keychain Manager. The Keychain Manager provides both high and low level functions for providing secure storage and transparent authentication to any application that uses AppleShare, Internet, or generic passwords.

Most applications only need to use high-level Keychain Manager functions, which enable you to store and retrieve passwords without requiring that the user unlock or create a new keychain. Low-level Keychain Manager functions require that you unlock a keychain in order to access keychain item data. To edit the contents of a keychain, it must be created with read/write access. You might wish to use low-level Keychain Manager functions if you are a network administrator writing a logging application that keeps track of network or system events.

The following sections provide an introduction to programming the Keychain Manager:

Determining Keychain Manager Availability and Version Information

You should call the `function KeychainManagerAvailable` (page 36) to determine whether the Keychain Manager is available before calling any other Keychain functions.

The function `KCGetKeychainManagerVersion` (page 36) returns a pointer to the version number of the Keychain Manager installed on the current system.

Creating a Keychain Reference

The Keychain Manager provides two functions you can use to create a keychain reference. The Keychain Manager uses a keychain reference to uniquely identify a keychain. The function `KCMakeKCRefFromFSSpec` (page 37) enables you to create a keychain reference from a file specification record. The function `KCMakeKCRefFromAlias` (page 38) enables you to create a keychain reference from a handle to an alias. You can obtain an alias handle to a keychain by calling the function `KCMakeAliasFromKCRef` (page 38). You may wish to call this function if you wish to determine the location of a keychain.

Your application should not assume a keychain is a file, because keychains may be stored on other media (such as a smart card) in future versions of Keychain Manager.

After you are finished with a keychain reference, you should call the function `KCReleaseKeychain` (page 39) to dispose of the memory it occupied. On return, `KCReleaseKeychain` sets the reference to `NULL`. You should not use it after this.

Creating a New Keychain

You can call the function `KCCreateKeychain` (page 39) to create a new keychain. `KCCreateKeychain` asks for a pointer to the password string that will be used to protect the new keychain, and a pointer to the keychain reference indicating where it should create the new keychain. You can pass `NULL` for both parameters. If you pass `NULL` for the password string, the Create Keychain dialog will be displayed to obtain it. In this case, the keychain is automatically unlocked after creation. If you pass a keychain reference whose location is unspecified or invalid, `KCCreateKeychain` creates the new keychain in the Keychains folder.

After you are finished with a keychain reference, you should call the function `KCReleaseKeychain` (page 39) to dispose of the memory it occupied. On return, `KCReleaseKeychain` sets the reference to `NULL`. You should not use it after this.

Locking and Unlocking a Keychain

The function `KCUnlock` (page 44) unlocks a keychain. In most cases, your application does not need to call the function `KCUnlock`, since most Keychain Manager functions requiring an unlocked keychain call `KCUnlock` automatically. `KCUnlock` may display the Unlock dialog if the keychain is currently locked. If your application needs to verify that a keychain is unlocked, call the function `KCGetStatus` (page 43).

You can call the function `KCLock` (page 45) to lock an unlocked keychain. Your application should not call `KCLock` unless you are directly responding to a user's request to lock a keychain. In general, you should leave the keychain unlocked so that the user does not have to unlock it again in another application.

Setting and Obtaining the Default Keychain

In most cases, your application should not set the default keychain. If you need to, call the function `KCSetDefaultKeychain` (page 41). You should call this function if you want to change where keychain items are added. If you indicate a locked keychain, the Unlock Keychain dialog will be displayed to prompt the user to unlock it.

You should call the function `KCGetDefaultKeychain` (page 41) to obtain a reference to the default keychain. You should call `KCGetDefaultKeychain` to obtain the name of the default keychain. To do so, pass the returned reference to the function `KCGetKeychainName` (page 43).

Setting and Retrieving Keychain Information

The Keychain Manager enables you to create and display dialogs that allow a user to set up and access their keychain, as well as to enable them to supply and modify passwords. It provides these dialogs to ensure user interaction in creating and opening keychains. For security reasons, a keychain cannot be unlocked programmatically. When the user clicks on the Keychain icon in the Control Strip, a dialog appears prompting the user to create a new keychain, unlock an existing keychain, or change the identity of an existing keychain.

You can call the function `KCChangeSettings` (page 42) to display a dialog that enables the user to change the name, password, and settings associated with a keychain. Pass `NULL` in the keychain parameter to indicate the default keychain.

The function `KCGetStatus` (page 43) retrieves the status of a keychain. It passes back a bit mask that you can test to determine the status of the specified keychain. See "Keychain Status Mask Constants" (page 106) for a description of this mask. The function `KCGetKeychainName` (page 43) retrieves the name of a keychain.

Searching Keychains

You can use the functions `KCCountKeychains` (page 46) and `KCGetIndKeychain` (page 46) to obtain a keychain reference corresponding to an indexed keychain. Pass a value between 1 and the number returned by the function `KCCountKeychains` in the index parameter of `KCGetIndKeychain`.

Storing and Retrieving Keychain Items

Prior to Keychain Manager 2.0, the user had to unlock a keychain before your application could determine whether a password was in the keychain. In 2.0 and later, when the Keychain Manager searches for a keychain item, it searches all unlocked keychains first, then searches all locked keychains. If the password is found in a locked keychain, the user will be prompted to unlock the keychain so the password can be retrieved. If it is not found in any keychain, the result code itemNotFound will be returned and user interaction will be required.

You can call the functions `KCAddAppleSharePassword` (page 48) and `KCFindAppleSharePassword` (page 49) to store and retrieve AppleShare passwords. To store Internet passwords, call the functions `KCAddInternetPassword` (page 52) and `KCAddInternetPasswordWithPath` (page 53). To retrieve them, call the functions `KCFindInternetPassword` (page 55) and `KCFindInternetPasswordWithPath` (page 57). Generic passwords are stored by the function `KCAddGenericPassword` (page 60) and retrieved by the function `KCFindGenericPassword` (page 61). Note that all of the functions listed above have alternate functions which perform the same task but take C strings instead of Pascal strings. See the function discussions for more information.

If you wish to select or retrieve certificates, you can call the functions `KCChooseCertificate` (page 77) and `KCFindX509Certificates` (page 76), respectively.

Listing 3-1 (page 28) demonstrates how your application could use these high-level Keychain Manager functions to store password data. Note that an explicit call to the function `KCUnlock` (page 44) to unlock the keychain is not required.

As illustrated, you should call the function `KeychainManagerAvailable` (page 36) before calling the rest of the API to determine whether the Keychain Manager is available. Your application must call the Memory Manager function `MaxApplZone` to utilize the maximum memory available.

Listing 3-1 Storing password data

```
OSStatus StorePasswordInKeychain (ConstStr255Param password)
{
    OSStatus status;

    if (!KeychainManagerAvailable ()) // is it there?
        return ((OSStatus) MY_ERROR);

    KCItemRef item;
    status = KCAddGenericPassword (
                 "\pMy_App_Pwd", // service name
                 "\pBill Braskey", // account name
                 password[0], // length of password
                 &password[1], // pointer to password data
                 &item);

    return (status);
}
```

Creating Keychain Item References

The Keychain Manager provides the function `KCNewItem` (page 63) to create a keychain item reference. A keychain item reference is a reference to an opaque structure that contains information about the keychain item. The Keychain Manager uses a keychain reference to uniquely identify a keychain item.

After you are finished with a keychain item reference, you should call the function `KCReleaseItem` (page 64) to dispose of the memory it occupied. On return, `KCReleaseItem` sets the reference to `NULL`. You should not use it after it has been released.

Setting and Obtaining Keychain Item Attribute Data

You can call the functions `KCSetAttribute` (page 65) and `KCSetData` (page 67) to set or modify attribute data for a keychain item. The difference between the two functions is that `KCSetData` requires that you pass the length of the data and a pointer to that data as separate parameters rather than fields in a keychain item attribute structure.

You can only set or modify standard item attributes identified by the tag constants `kDescriptionKCItemAttr`, `kCommentKCItemAttr`, `kLabelKCItemAttr`, `kCreatorKCItemAttr`, `kTypeKCItemAttr`, and `kCustomIconKCItemAttr`. In addition, each class of keychain item has attributes specific to that class which may be set or modified. See "Keychain Item Attribute Tag Constants" (page 99) for a description of item attribute tags.

You can call the functions `KCGetAttribute` (page 66) and `KCGetData` (page 68) to obtain keychain item attribute data. The difference between the two functions is that `KCGetAttribute` requires that you pass the length of the data buffer as a field in a keychain item attribute structure rather than as a separate parameter. It passes a pointer to the attribute structure rather than a pointer to the attribute data and the actual length of that data as separate parameters.

You can only obtain standard item attributes identified by the tag constants `kClassKCItemAttr`, `kCreationDateKCItemAttr`, `kModDateKCItemAttr`, `kDescriptionKCItemAttr`, `kCommentKCItemAttr`, `kLabelKCItemAttr`, `kCreatorKCItemAttr`, `kScriptCodeKCItemAttr`, and `kCustomIconKCItemAttr`. In addition, each class of keychain item has attributes specific to that class which may be obtained. See "Keychain Item Attribute Tag Constants" (page 99) for a description of item attribute tags.

Before calling these functions to access keychain item data, you must call the function `KCUnlock` (page 44) to unlock the keychain. Note that a keychain must allow read/write access for you to edit its contents.

Manipulating Keychain Items

You can call the functions `KCAddItem` (page 70) and `KCDeleteItem` (page 70) to add and delete keychain items from the default keychain. After calling `KCDeleteItem`, you should call the function `KCReleaseItem` (page 64) when you are finished with an item, since `KCDeleteItem` does not dispose the memory occupied by the item reference.

You can call the function `KCUpdateItem` (page 71) to update an item after changing its attributes or data. The item is written to the keychain's permanent data store. `KCUpdateItem` may display the Unlock Keychain dialog if the keychain containing the item is currently locked.

You can use the function `KCCopyItem` (page 72) to copy a keychain item from one keychain to another. The function `KCGetKeychain` (page 72) retrieves the location of a keychain item.

Searching for Keychain Items

You can use the function `KCFindFirstItem` (page 74) to find the first the first keychain item in a keychain that matches certain attributes. `KCFindFirstItem` passes back a reference to the item and to the current search criteria. You pass the search reference returned by to `KCFindFirstItem` to the function `KCFindNextItem` (page 75). `KCFindNextItem` finds the next keychain item matching the criteria used by `KCFindFirstItem` and returns a reference to the matching item, if any.

When you are completely finished with a search performed by calling the functions `KCFindNextItem` (page 75) or `KCFindNextItem` (page 75), call the function `KCReleaseSearch` (page 76) to release the memory occupied by a search criteria reference.

Working With Certificates

The Keychain Manager provides two functions to give you limited access to certificates. The function `KCFindX509Certificates` (page 76)finds certificates in a keychain that match the given search criteria. You set these criteria by passing a bit mask in the options parameter. This mask is described in "Certificate Search Option Mask Constants" (page 92). The function `KCChooseCertificate` (page 77) displays a list of certificates that the user can choose from. If only one certificate is available matching the criteria, `KCChooseCertificate` does not present a user interface, and instead passes back the certificate reference in the `certificate` parameter.

Note that both certificate functions are supported on Mac OS 9, but will not be supported in the first release of OS X.

Managing User Interaction

You can call the function `KCSetInteractionAllowed` (page 79) to tell Keychain Manager functions that display user interface whether to do so. If you pass `true` in the state parameter, user interaction is allowed. If you pass `false,` user interaction is not allowed. In this case, Keychain functions that normally display a user interface will instead return an error. Failing to re-enable user interaction will affect other clients of the Keychain Manager. The interaction allowed state is reset when the machine reboots to the default state, that is, user interaction allowed.

The function `KCIsInteractionAllowed` (page 79) returns a `Boolean` value indicating whether Keychain Manager functions that display user interface will do so. If `true`, user interaction is allowed, and Keychain Manager is free to show a user interface when needed.

Responding to Keychain Events

A keychain event is generated when one of the following situations occurs:

- The lock state of a keychain changes
- The settings of a keychain change
- The default keychain changes
- A keychain item is found
- A system event occurs while your application wants to update its windows
- Data is read from a item keychain item
- A keychain item is added, updated, or removed
- A keychain call takes more than five ticks (at 60 ticks per second) to complete

The simplest approach to handling keychain events is to respond to the event `kSystemKCEvent` when the Keychain Manager causes an update event to occur for your application's user interface. Applications that are interested in receiving keychain events must first call the function `KCAddCallback` (page 80) to register a callback that handles the event. You indicate the events you want to receive in the `eventMask` parameter of this function. Once you have registered your function, the Keychain Manager will invoke your callback when the specified keychain event(s) occur. When you no longer want to handle keychain events, call the function `KCRemoveCallback` (page 81) to unregister your callback function.

You may wish to write your own callback function to enable the dialogs displayed by Keychain Manager to be movable and resizable. Set the bit specified by the mask constant `kSystemKCEvent` in the `eventMask` parameter of the function `KCAddCallback` (page 80) to ensure that window updating occurs correctly.

All keychain API calls are synchronous. They do not return until complete. If called within a thread, this is not normally an issue. However, if your application is not threaded, you may wish to write your own callback function to handle keychain idle events. Your callback function will be called periodically until the function completes. Your application must call the functions `YieldToAnyThread` or `WaitNextEvent` when an idle event is generated. If your callback does not specify that it will handle idle events, the Keychain Manager will periodically call `YieldToAnyThread`.

When a keychain event occurs, the Keychain Shared Library calls your callback function. Allocating memory or calling Mac OS Toolbox functions that do so is unsafe in this circumstance, unless the event is an idle event. When an idle event occurs, your application is free to make Toolbox calls and perform memory allocation.

Listing 3-2 (page 34) shows how your application might register your callback function to handle keychain events. Listing 3-3 (page 34) illustrates a sample keychain event handling callback function.

Listing 3-2 Registering your callback function

```
OSStatus RegisterMyCallBackProc (Ptr myDataPtr)
{
    OSStatus status = noErr;
    static KCCallbackUPP myCallbackUPP = nil;

    if (!myCallbackUPP)
    {
        // create a UPP for callback function
        myCallbackUPP = NewKCCallbackProc(MyCallbackProc);
    }
    status = KCAddCallback(myCallbackUPP, kcEveryEvent, myDataPtr);
    return (status);
}
```

Listing 3-3 Creating your own callback function

```
pascal void MyCallbackProc (
                KCEvent inEvent,
                KCCallbackInfo *info,
                void *userContext)
{
    MyDataPtrType myDataPtr = (MyDataPtrType) userContext;
    if (inEvent == kcIdleEvent)
    {
        YieldToAnyThread();
    }
    else if (myDataPtr != nil)
    {
        // it may not be safe to allocate or move memory here,
        // so you may want to queue the event for later processing
        myDataPtr->event = inEvent;
        myDataPtr->item = inInfo->Item;
        myDataPtr->gotAnEvent = True;
    }
}
```

C H A P T E R 4

Keychain Manager Reference

The following sections provide a complete description of the Keychain 2.0 API, including its functions, data types, constants, and result codes.

Keychain Manager Functions

Determining Availability and Version Information

`KeychainManagerAvailable` — Determines whether the Keychain Manager is available. (page 36)

`KCGetKeychainManagerVersion` — Determines the version of the Keychain Manager installed on the user's system. (page 36)

KeychainManagerAvailable

Determines whether the Keychain Manager is available.

```
Boolean KeychainManagerAvailable (void);
```

function result A `Boolean` value indicating whether the Keychain Manager is available. If `true`, your application can call Keychain Manager functions.

Discussion
You should call the `KeychainManagerAvailable` function to determine whether the Keychain Manager is available before calling any other Keychain functions.

Version Notes
Available beginning with Keychain Manager 1.0.

KCGetKeychainManagerVersion

Determines the version of the Keychain Manager installed on the user's system.

```
OSStatus KCGetKeychainManagerVersion (UInt32 *returnVers);
```

`returnVers`

On return, a pointer to the version number of the Keychain Manager installed on the current system.

Discussion

Your application can call the `KCGetKeychainManagerVersion` function to find out which version of the Keychain Manager is installed on the user's system.

Version Notes

Available beginning with Keychain Manager 1.0.

Creating a Keychain Reference

`KCMakeKCRefFromFSSpec` — Creates a keychain reference from a file specification record. (page 37)

`KCMakeKCRefFromAlias` — Creates a keychain reference from a keychain alias. (page 38)

`KCMakeAliasFromKCRef` — Creates an alias to a keychain reference. (page 38)

`KCReleaseKeychain` — Disposes of the memory associated with a keychain reference. (page 39)

KCMakeKCRefFromFSSpec

Creates a keychain reference from a file specification record.

```
OSStatus KCMakeKCRefFromFSSpec (
                FSSpec *keychainFSSpec,
                KCRef *keychain);
```

`keychainFSSpec`

A pointer to a keychain file specification record.

`keychain`

On return, a pointer to a reference to the keychain specified by the file in the `keychainFSSpec` parameter.

Version Notes

Available beginning with Keychain Manager 2.0.

Special Considerations

The memory that the keychain reference occupies must be released by calling the function `KCReleaseKeychain` (page 39) when you are finished with it. You should not use the reference after it has been released.

KCMakeKCRefFromAlias

Creates a keychain reference from a keychain alias.

```
OSStatus KCMakeKCRefFromAlias(
                    AliasHandle keychainAlias,
                    KCRef *keychain);
```

`keychainAlias`
: A handle to an alias record of the keychain file. Since the keychain is a file, an alias can be made to the keychain file.

`keychain`
: On return, a pointer to a reference to the keychain specified by the alias in the `keychainAlias` parameter.

Special Considerations

The memory that the keychain reference occupies must be released by calling the function `KCReleaseKeychain` (page 39) when you are finished with it. You should not use the reference after it has been released.

KCMakeAliasFromKCRef

Creates an alias to a keychain reference.

```
OSStatus KCMakeAliasFromKCRef(
                    KCRef keychain,
                    AliasHandle *keychainAlias);
```

`keychain`
: A reference to the keychain for which you want to create an alias.

`AliasHandle`
: On return, a pointer to an alias handle to the file referred to by the keychain reference.

Discussion

You may wish to call the `KCMakeAliasFromKCRef` function to determine the location of a keychain.

Version Notes
Available beginning with Keychain Manager 2.0.

Special Considerations
The memory that the keychain reference occupies must be released by calling the function `KCReleaseKeychain` (page 39) when you are finished with it.

KCReleaseKeychain

Disposes of the memory associated with a keychain reference.

```
OSStatus KCReleaseKeychain (KCRef *keychain);
```

`keychain`
: A pointer to a keychain reference. Pass the keychain reference whose memory you want to release. On return, the reference is set to `NULL` and should not be used again.

Discussion
You should call the `KCReleaseKeychain` function to release the memory occupied by a keychain reference when you are finished with it. You should not use the reference after it has been released.

Version Notes
Available beginning with Keychain Manager 1.0.

Creating a New Keychain

`KCCreateKeychain` — Creates a new keychain. (page 39)

KCCreateKeychain

Creates a new keychain.

```
OSStatus KCCreateKeychain (
                    StringPtr password,
                    KCRef *keychain);
```

`password`
: A pointer to a Pascal string representing the password string which will be used to protect the new keychain. If you pass `NULL`, the Keychain Setup dialog will be displayed to obtain it.

`keychain`
: A pointer to a reference to the keychain you wish to create. You create a reference by calling the function `KCMakeKCRefFromFSSpec` (page 37). If you pass a `NULL` pointer, `KCCreateKeychain` creates the new keychain in the Keychains folder. If you pass a valid reference pointing to `NULL`, the Keychain Manager allocates the memory for the keychain reference and returns it in this parameter. In this case, the keychain will be created in the Keychain folder.

function result The result code `userCanceledErr` indicates that the user pressed the Cancel button in the create keychain. The result code `errKCDuplicateKeychain` indicates that the user tried to create a keychain which already exists. The result code `errKCInvalidKeychain` indicates that the specified keychain is invalid. Additional errors may be returned if the keychain could not be created (for example, a file system or network error may be returned if there is no write access to the storage media).

Discussion

The `KCCreateKeychain` function creates a new empty keychain optionally defined by the `keychain` and `password` parameters. If the specified keychain reference is `NULL` or points to `NULL`, the new keychain will be created in the Keychains folder within the Preferences folder of the System folder. If you pass a keychain reference whose location is unspecified or invalid, the Keychain Manager will fill in the location to the Keychains folder. If the Keychain Manager posts a user interface to create a keychain, that keychain is automatically unlocked after creation.

You can also call the function `kccreatekeychain` to create an empty keychain. The difference between the two functions is that kccreatekeychain takes a pointer to a C string instead of a Pascal string in the `password` parameter.

Version Notes

Available beginning with Keychain Manager 1.0.

Special Considerations

The memory that the keychain reference occupies must be released by calling the function `KCReleaseKeychain` (page 39) when you are finished with it.

Setting and Obtaining the Default Keychain

`KCSetDefaultKeychain` — Sets the default keychain. (page 41)

`KCGetDefaultKeychain` — Obtains the default keychain. (page 41)

KCSetDefaultKeychain

Sets the default keychain.

```
OSStatus KCSetDefaultKeychain (KCRef keychain);
```

`keychain`	A reference to a keychain that you want to make the default.
function result	The result code `errKCNoSuchKeychain` indicates that the specified keychain could not be found. The result code `errKCInvalidKeychain` indicates that the specified keychain is invalid.

Discussion
In most cases, your application should not set the default keychain. If you need to, call the `KCSetDefaultKeychain` function. You should call this function if you want to change where keychain items are added. If you indicate a locked keychain, the Unlock Keychain dialog will be displayed to prompt the user to unlock it.

Version Notes
Available beginning with Keychain Manager 1.0.

KCGetDefaultKeychain

Obtains the default keychain.

```
OSStatus KCGetDefaultKeychain (KCRef *keychain);
```

`keychain`	On return, a pointer to a reference to the default keychain.
function result	The result code `errKCNoDefaultKeychain` indicates that there is no default keychain.

Discussion

You should call the `KCGetDefaultKeychain` function to obtain a reference to the default keychain. You should call `KCGetDefaultKeychain` to obtain the name of the default keychain. To do so, pass the returned reference to the function `KCGetKeychainName` (page 43).

Version Notes

Available beginning with Keychain Manager 1.0.

Setting and Obtaining Keychain Information

`KCChangeSettings` — Displays a dialog for changing the keychain name, password, and settings associated with a specified keychain. (page 42)

`KCGetStatus` — Obtains the status of a keychain. (page 43)

`KCGetKeychainName` — Obtains the name of a keychain. (page 43)

`KCGetKeychain` — Retrieves the location of a keychain item. (page 72)

KCChangeSettings

Displays a dialog for changing the keychain name, password, and settings associated with a specified keychain.

```
OSStatus KCChangeSettings (KCRef keychain);
```

`keychain`	A reference to an unlocked keychain. Pass in `NULL` to specify the default keychain.
function result	The result code `errUserCanceled` indicates that the user pressed the Cancel button in the Change Settings dialog. The result code `errKCNoDefaultKeychain` indicates that the default keychain could not be found. The result code `errKCInvalidKeychain` indicates that the specified keychain is invalid.

Discussion

You application should not normally call the `KCChangeSettings` function, unless responding directly to a user's request to change keychain settings. An application cannot directly change a keychain's passphrase. For security reasons, your application can effect a change to the passphrase only by calling `KCChangeSettings` to allow the user to change the passphrase interactively.

Version Notes

Available beginning with Keychain Manager 1.0.

KCGetStatus

Obtains the status of a keychain.

```
OSStatus KCGetStatus (
                    KCRef *keychain,
                    UInt32 *keychainStatus);
```

`keychain`
: A pointer to the keychain reference whose status you wish to determine. Pass `NULL` to obtain the status of the default keychain.

`keychainStatus`
: On return, a pointer to a bitmask that you can test to determine the status of the specified keychain. See "Keychain Status Mask Constants" (page 106) for a description of this mask.

function result The result code `errKCNoSuchKeychain` indicates that the specified keychain could not be found. The result code `errKCInvalidKeychain` indicates that the specified keychain is invalid.

Version Notes

Available beginning with Keychain Manager 1.0.

KCGetKeychainName

Obtains the name of a keychain.

```
OSStatus KCGetKeychainName (
                    KCRef keychain,
                    StringPtr keychainName);
```

`keychain`
: A reference to the keychain whose name you wish to obtain.

`keychainName`
: A pointer to a Pascal string. On return, this string contains the name of the keychain.

function result The result code `errKCInvalidKeychain` indicates that the keychain is invalid.

Discussion

You can also call the function `kcgetkeychainname` to obtain the name of a keychain. The difference between the two functions is that `kcgetkeychainname` takes a pointer to a C string instead of a Pascal string in the `keychainName` parameter.

Version Notes

Available beginning with Keychain Manager 2.0.

Locking and Unlocking Keychains

`KCUnlock` — Unlocks a keychain. (page 44)

`KCLock` — Locks a keychain. (page 45)

KCUnlock

Unlocks a keychain.

```
 OSStatus KCUnlock (
                    KCRef keychain,
                    StringPtr password);
```

`keychain`
: A reference to the keychain to unlock. Pass `NULL` to specify the default keychain. If you pass `NULL` and the default keychain is currently locked, the keychain will appear as the default choice. If you pass a locked keychain, `KCUnlock` will display the Unlock Keychain dialog and the keychain will appear as the chosen menu item in keychain popup menu. If the default keychain is currently unlocked, the Unlock Keychain dialog is not displayed and `KCUnlock` returns `noErr`.

`password`
: A pointer to a Pascal string representing the password string for this keychain. Pass `NULL` if the user password is unknown. In this case, `KCUnlock` displays the Unlock Keychain dialog, and the

authentication user interface associated with the keychain about to be unlocked. If you specify an invalid password, you will not be able to unlock the keychain with a specified password until the machine is rebooted. In this case, `KCUnlock` returns `errKCInteractionRequired`.

function result The result code `noErr` does not guarantee that the specified keychain is unlocked, because the user can select any available keychain and unlock it. The result code `errUserCanceled` indicates that the user pressed the Cancel button in the Unlock Keychain dialog. The result code `errKCAuthFailed` indicates that authentication failed because of too many unsuccessful retries. The result code `errKCInteractionRequired` indicates that user interaction is required to unlock the keychain. In this case, you will not be able to unlock the keychain with that password until the machine is rebooted.

Discussion

In most cases, your application does not need to call the `KCUnlock` function directly, since most Keychain Manager functions requiring an unlocked keychain call `KCUnlock` automatically. `KCUnlock` may display the Unlock Keychain dialog if the keychain is currently locked. If your application needs to verify that a keychain is unlocked, call the function `KCGetStatus` (page 43). `KCUnlock` replaces the function `KCUnlockKeychain`, which was available in Keychain Manager 1.0.

You can also call the function `kcunlock` to unlock a keychain. The difference between the two functions is that `kcunlock` takes a pointer to a C string instead of a Pascal string in the `password` parameter.

Version Notes

Available beginning with Keychain Manager 2.0.

Special Considerations

The memory that the keychain reference occupies must be released by calling the function `KCReleaseKeychain` (page 39) when you are finished with it.

KCLock

Locks a keychain.

```
OSStatus KCLock (KCRef keychain);
```

`keychain`

A reference to the keychain to lock. Pass `NULL` to lock all unlocked keychains.

function result The result code `errKCNoSuchKeychain` indicates that specified keychain could not be found. The result code `errKCInvalidKeychain` indicates that the specified keychain is invalid.

Discussion

The `KCLock` function locks a keychain if it is unlocked. Your application should not call `KCLock` unless you are directly responding to a user's request to lock a keychain. In general, you should leave the keychain unlocked so that the user does not have to unlock it again in another application. `KCLock` replaces the function `KCLockKeychain`, which was available in Keychain Manager 1.0.

Version Notes

Available beginning with Keychain Manager 1.0.

Searching for Keychains

`KCCountKeychains` — Determines the number of available keychains. (page 46)

`KCGetIndKeychain` — Obtains an indexed keychain reference. (page 46)

KCCountKeychains

Determines the number of available keychains.

```
UInt16 KCCountKeychains (void);
```

function result The number of available keychains. This includes all keychains in the Keychains folder, as well as any other keychains known to the Keychain Manager.

Version Notes

Available beginning with Keychain Manager 1.0.

KCGetIndKeychain

Obtains an indexed keychain reference.

```
OSStatus KCGetIndKeychain (
                    UInt16 index,
                    KCRef *keychain);
```

`index`
: An index of the list of available keychains. Pass a value between 1 and the number returned by the function `KCCountKeychains` (page 46).

`keychain`
: A pointer to the keychain reference corresponding to the index in the `index` parameter.

function result The result code `errKCNoSuchKeychain` indicates that the index value is out of range.

Version Notes
Available beginning with Keychain Manager 1.0.

Special Considerations
The memory that the keychain reference occupies must be released by calling the function `KCReleaseKeychain` (page 39) when you are finished with it.

Storing and Retrieving Passwords

`KCAddAppleSharePassword` — Adds a new AppleShare server password to the default keychain. (page 48)

`KCFindAppleSharePassword` — Finds the first AppleShare password in the default keychain that matches the specified parameters. (page 49)

`KCAddInternetPassword` — Adds a new Internet server password to the default keychain. (page 52)

`KCAddInternetPasswordWithPath` — Adds a new Internet server password with a specified path to the default keychain. (page 53)

`KCFindInternetPassword` — Finds the first Internet password in the default keychain that matches the specified parameters. (page 55)

`KCFindInternetPasswordWithPath` — Finds the first Internet password in the default keychain that matches the specified parameters, including path information. (page 57)

`KCAddGenericPassword` — Adds a new generic password to the default keychain. (page 60)

`KCFindGenericPassword` — Finds the first generic password in the default keychain matching specified parameters. (page 61)

KCAddAppleSharePassword

Adds a new AppleShare server password to the default keychain.

```
OSStatus KCAddAppleSharePassword(
                    AFPServerSignature *serverSignature,
                    StringPtr serverAddress,
                    StringPtr serverName,
                    StringPtr volumeName,
                    StringPtr accountName,
                    UInt32 passwordLength,
                    const void *passwordData,
                    KCItemRef *item);
```

`serverSignature`
: A pointer to a 16-byte Apple File Protocol server signature block. Pass a value of type `AFPServerSignature` (page 86). Pass `NULL` to match any server signature. The Keychain Manager identifies the location for the password by the information passed in the `serverAddress` and `serverSignature` parameters. You must pass a valid value in at least one of these parameters.

`serverAddress`
: A pointer to a Pascal string containing the server address, which may be specified as an AppleTalk zone name, a DNS domain name (in the format "xxx.yyy.zzz"), or an IP address (in the format "111.222.333.444"). The Keychain Manager identifies the location for the password by the information passed in the `serverAddress` and `serverSignature` parameters. You must pass a valid value in at least one of these parameters.

`serverName`
: A pointer to a Pascal string containing the server name.

`volumeName`
: A pointer to a Pascal string containing the volume name.

`accountName`
: A pointer to a Pascal string containing the account name.

`passwordLength`
: The length of the buffer pointed to by `passwordData`.

`passwordData`
: A pointer to a buffer which will hold the returned password data. Before calling `KCAddAppleSharePassword`, allocate enough memory for the buffer to hold the data you want to store.

`item`
: On return, a pointer to a reference to the added item. Pass `NULL` if you don't want to obtain this reference.

function result
: The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCDuplicateItem` indicates that you tried to add a password that already exists in the keychain. The result code `errKCDataTooLarge` indicates that you tried to add more data than is allowed for a record of this type.

Discussion

The `KCAddAppleSharePassword` function adds a new AppleShare server password to the default keychain that is uniquely identified by the `serverName`, `volumeName`, `accountName` parameters, and a location specified either by `serverAddress` or `serverSignature`. `KCAddAppleSharePassword` optionally returns a reference to the newly added item.

Most applications do not need to store AppleShare password data, as this is handled transparently by the AppleShare client software. To be compatible with the AppleShare client, you should store a fully-specified File Manager structure `AFPXVolMountInfo` as the password data.

You can also call the function `kcaddapplesharepassword` to add an AppleShare server password to the default keychain. The difference between the two functions is that `kcaddapplesharepassword` takes a pointer to a C string instead of a Pascal string in the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes

Available beginning with Keychain Manager 1.0.

KCFindAppleSharePassword

Finds the first AppleShare password in the default keychain that matches the specified parameters.

```
OSStatus KCFindAppleSharePassword(
                    AFPServerSignature *serverSignature,
                    StringPtr serverAddress,
                    StringPtr serverName,
                    StringPtr volumeName,
                    StringPtr accountName,
                    UInt32 maxLength,
                    void *passwordData,
                    UInt32 *actualLength,
                    KCItemRef *item);
```

`serverSignature`
: A pointer to a 16-byte Apple File Protocol server signature block. Pass a value of type `AFPServerSignature` (page 86). Pass `NULL` to match any server signature. The Keychain Manager identifies the location for the password by the information passed in the `serverAddress` and `serverSignature` parameters. You must pass a valid value in at least one of these parameters.

`serverAddress`
: A pointer to a Pascal string containing the server address, which may be specified as an AppleTalk zone name, a DNS domain name (in the format "xxx.yyy.zzz"), or an IP address (in the format "111.222.333.444"). The Keychain Manager identifies the location for the password by the information passed in the `serverAddress` and `serverSignature` parameters. You must pass a valid value in at least one of these parameters.

`serverName`
: A pointer to a Pascal string containing the server name. Pass `NULL` to match any server name.

`volumeName`
: A pointer to a Pascal string containing the volume name. Pass `NULL` to match any volume name.

`accountName`
: A pointer to a Pascal string containing the account name. Pass `NULL` to match any account name.

`maxLength`
: The length of the buffer pointed to by `passwordData`. Pass 0 if you want to obtain the item reference but not the password data. In this case, you must also pass `NULL` in the `passwordData` parameter.

`passwordData`
: A pointer to a buffer which will hold the returned password data. Before calling `KCFindAppleSharePassword`, allocate enough memory for the buffer to hold the data you want to store. Pass `NULL` if you want to obtain the item reference but not the password data. In this case, you must also pass 0 in the `maxLength` parameter. On return, a pointer to the returned password data.

`actualLength`
On return, the actual length of the password data that was retrieved. If the buffer pointed to by `passwordData` is smaller than the actual length of the data, `KCFindAppleSharePassword` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling `KCFindAppleSharePassword` again.

`item`
On return, a pointer to a reference to the found item. Pass `NULL` if you don't want to obtain this reference.

function result The result code `errKCNoDefaultKeychain` indicates that no default keychain was found. The result code `errKCItemNotFound` indicates that no matching password item was found. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling `KCFindAppleSharePassword` again.

Discussion

The `KCFindAppleSharePassword` function finds the first AppleShare password item which matches the attributes you provide. The buffer specified in the `passwordData` parameter must be large enough to hold the password data, otherwise `KCFindAppleSharePassword` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling `KCFindAppleSharePassword` again. `KCFindAppleSharePassword` optionally returns a reference to the found item.

You can also call the function `kcfindapplesharepassword` to find the first AppleShare server password matching specified attributes. The difference between the two functions is that `kcfindapplesharepassword` takes a pointer to a C string instead of a Pascal string in the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes

Available beginning with Keychain Manager 1.0.

KCAddInternetPassword

Adds a new Internet server password to the default keychain.

```
OSStatus KCAddInternetPassword (
                    StringPtr serverName,
                    StringPtr securityDomain,
                    StringPtr accountName,
                    UInt16 port,
                    OSType protocol,
                    OSType authType,
                    UInt32 passwordLength,
                    const void *passwordData,
                    KCItemRef *item);
```

`serverName`
: A pointer to a Pascal string containing the server name.

`securityDomain`
: A pointer to a Pascal string containing the security domain. This parameter is optional, as not all protocols will require it.

`accountName`
: A pointer to a Pascal string containing the account name.

`port`
: The TCP/IP port number. Pass the constant `kAnyPort`, described in "Default Port Constant" (page 95), to specify any port.

`protocol`
: The protocol associated with this password. See "Keychain Protocol Type Constants" (page 105) for a description of possible values. Pass the constant `kAnyProtocol`, described in "Default Protocol Constant" (page 95), to specify any protocol.

`authType`
: The authentication scheme used. See "Authentication Type Constants" (page 91) for a description of possible values. Pass the constant `kAnyAuthType`, described in "Default Authentication Type Constant" (page 94), to specify any authentication scheme.

`passwordLength`
: The length of the buffer pointed to by `passwordData`.

`passwordData`
: A pointer to a buffer which will hold the returned password data. Before calling `KCAddInternetPasswordWithPath`, allocate enough memory for the buffer to hold the data you want to store.

`item`

On return, a pointer to a reference to the added item. Pass `NULL` if you don't want to obtain this reference.

function result The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCDuplicateItem` indicates that you tried to add a password that already exists in the keychain. The result code `errKCDataTooLarge` indicates that you tried to add more data than is allowed for a record of this type.

Discussion

The `KCAddInternetPassword` function adds a new Internet server password to the default keychain. Required parameters to identify the password are `serviceName` and `accountName` (you cannot pass `NULL` for both parameters). In addition, some protocols may require an optional `securityDomain` when authentication is requested. `KCAddInternetPassword` optionally returns a reference to the newly added item.

You can also call the function `kcaddinternetpassword` to add a new Internet server password to the default keychain. The difference between the two functions is that `kcaddinternetpassword` takes a pointer to a C string instead of a Pascal string in the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes

Available beginning with Keychain Manager 1.0.

KCAddInternetPasswordWithPath

Adds a new Internet server password with a specified path to the default keychain.

```
OSStatus KCAddInternetPasswordWithPath (
                    StringPtr serverName,
                    StringPtr securityDomain,
                    StringPtr accountName,
                    StringPtr path,
                    UInt16 port,
                    OSType protocol,
                    OSType authType,
                    UInt32 passwordLength,
                    const void *passwordData,
                    KCItemRef *item);
```

`serverName`

A pointer to a Pascal string containing the server name.

`securityDomain`
: A pointer to a Pascal string containing the security domain. This parameter is optional, as not all protocols will require it.

`accountName`
: A pointer to a Pascal string containing the account name.

`path`
: A pointer to a Pascal string containing additional information that specifies a file or directory on the server specified by `serverName`. In a typical URL, path information begins directly after the first slash ("/") character following the server name. This parameter is optional.

`port`
: The TCP/IP port number. Pass the constant `kAnyPort`, described in "Default Port Constant" (page 95), to specify any port.

`protocol`
: The protocol associated with this password. See "Keychain Protocol Type Constants" (page 105) for a description of possible values. Pass the constant `kAnyProtocol`, described in "Default Protocol Constant" (page 95), to specify any protocol.

`authType`
: The authentication scheme used. See "Authentication Type Constants" (page 91) for a description of possible values. Pass the constant `kAnyAuthType`, described in "Default Authentication Type Constant" (page 94), to specify any authentication scheme.

`passwordLength`
: The length of the buffer pointed to by `passwordData`.

`passwordData`
: A pointer to a buffer which will hold the returned password data. Before calling `KCAddInternetPasswordWithPath`, allocate enough memory for the buffer to hold the data you want to store.

`item`
: On return, a pointer to a reference to the added item. Pass `NULL` if you don't want to obtain this reference.

function result The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCDuplicateItem` indicates that you tried to add a password that already exists in the keychain. The result code `errKCDataTooLarge` indicates that you tried to add more data than is allowed for a record of this type.

Discussion

The KCAddInternetPasswordWithPath function enables you to specify path information when adding a new Internet server password to the default keychain. Required parameters to identify the password are serviceName and accountName (you cannot pass NULL for both parameters). In addition, some protocols may require an optional securityDomain when authentication is requested. KCAddInternetPasswordWithPath optionally returns a reference to the newly added item.

You can also call the function kcaddinternetpasswordwithpath to add a new Internet server password to the default keychain. The difference between the two functions is that kcaddinternetpasswordwithpath takes a pointer to a C string instead of a Pascal string in the serverAddress, serverName, volumeName, accountName, and passwordData parameters.

Version Notes

Available beginning with Keychain Manager 2.0.

KCFindInternetPassword

Finds the first Internet password in the default keychain that matches the specified parameters.

```
OSStatus KCFindInternetPassword (
                    StringPtr serverName,
                    StringPtr securityDomain,
                    StringPtr accountName,
                    UInt16 port,
                    OSType protocol,
                    OSType authType,
                    UInt32 maxLength,
                    void *passwordData,
                    UInt32 *actualLength,
                    KCItemRef *item);
```

serverName
: A pointer to a Pascal string containing the server name. Pass NULL to match any server name.

securityDomain
: A pointer to a Pascal string containing the security domain. Pass NULL to match any domain.

`accountName`
: A pointer to a Pascal string containing the account name. Pass `NULL` to match any account name.

`port`
: The TCP/IP port number. Pass the constant `kAnyPort`, described in "Default Port Constant" (page 95), to match any port.

`protocol`
: The protocol associated with this password. See "Keychain Protocol Type Constants" (page 105) for a description of possible values. Pass the constant `kAnyProtocol`, described in "Default Protocol Constant" (page 95), to match any protocol.

`authType`
: The authentication scheme used. See "Authentication Type Constants" (page 91) for a description of possible values. Pass the constant `kAnyAuthType`, described in "Default Authentication Type Constant" (page 94), to match any authentication scheme.

`maxLength`
: The length of the buffer pointed to by `passwordData`. Pass 0 if you want to obtain the item reference but not the password data. In this case, you must also pass `NULL` in the `passwordData` parameter.

`passwordData`
: A pointer to a buffer which will hold the returned password data. Before calling `KCFindInternetPassword`, allocate enough memory for the buffer to hold the data you want to store. Pass `NULL` if you want to obtain the item reference but not the password data. In this case, you must also pass 0 in the `maxLength` parameter. On return, a pointer to the returned password data.

`actualLength`
: On return, the actual length of the password data that was retrieved. If the buffer pointed to by `passwordData` is smaller than the actual length of the data, `KCFindInternetPassword` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling `KCFindInternetPassword` again.

`item`
: On return, a pointer to a reference to the found item. Pass `NULL` if you don't want to obtain this reference.

function result The result code errKCNoDefaultKeychain indicates that no default keychain was found. The result code errKCItemNotFound indicates that no matching password item was found. The result code errKCBufferTooSmall indicates that your application must allocate a new buffer of sufficient size before calling KCFindInternetPassword again.

Discussion
The KCFindInternetPassword function finds the first Internet password item which matches the attributes you provide. The buffer specified in the passwordData parameter must be large enough to hold the password data, otherwise KCFindInternetPassword returns the result code errKCBufferTooSmall. In this case, your application must allocate a new buffer of sufficient size before calling KCFindInternetPassword again. KCFindInternetPassword optionally returns a reference to the found item.

You can also call the function kcfindinternetpassword to find the first Internet password item matching specified attributes. The difference between the two functions is that kcfindinternetpassword takes a pointer to a C string instead of a Pascal string in the serverAddress, serverName, volumeName, accountName, and passwordData parameters.

Version Notes
Available beginning with Keychain Manager 1.0.

KCFindInternetPasswordWithPath

Finds the first Internet password in the default keychain that matches the specified parameters, including path information.

```
OSStatus KCFindInternetPasswordWithPath (
                    StringPtr serverName,
                    StringPtr securityDomain,
                    StringPtr accountName,
                    StringPtr path,
                    UInt16 port,
                    OSType protocol,
                    OSType authType,
                    UInt32 maxLength,
                    void *passwordData,
                    UInt32 *actualLength,
                    KCItemRef *item);
```

`serverName`
: A pointer to a Pascal string containing the server name. Pass `NULL` to match any server name.

`securityDomain`
: A pointer to a Pascal string containing the security domain. Pass `NULL` to match any domain.

`accountName`
: A pointer to a Pascal string containing the account name. Pass `NULL` to match any account name.

`path`
: A pointer to a Pascal string containing additional information that specifies a file or directory on the server specified by `serverName`. In a typical URL, path information begins directly after the first slash ("/") character following the server name. This parameter is optional.

`port`
: The TCP/IP port number. Pass the constant `kAnyPort`, described in "Default Port Constant" (page 95), to match any port.

`protocol`
: The protocol associated with this password. See "Keychain Protocol Type Constants" (page 105) for a description of possible values. Pass the constant `kAnyProtocol`, described in "Default Protocol Constant" (page 95), to match any protocol.

`authType`
: The authentication scheme used. See "Authentication Type Constants" (page 91) for a description of possible values. Pass the constant `kAnyAuthType`, described in "Default Authentication Type Constant" (page 94), to match any authentication scheme.

`maxLength`
: The length of the buffer pointed to by `passwordData`. Pass 0 if you want to obtain the item reference but not the password data. In this case, you must also pass `NULL` in the `passwordData` parameter.

`passwordData`
: A pointer to a buffer which will hold the returned password data. Before calling `KCFindInternetPasswordWithPath`, allocate enough memory for the buffer to hold the data you want to store. Pass `NULL` if you want to obtain the item reference but not the password data. In this case, you must also pass 0 in the `maxLength` parameter. On return, a pointer to the returned password data.

`actualLength`

On return, the actual length of the password data that was retrieved. If the buffer pointed to by `passwordData` is smaller than the actual length of the data, `KCFindInternetPasswordWithPath` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling `KCFindInternetPasswordWithPath` again.

`item`

On return, a pointer to a reference to the found item. Pass `NULL` if you don't want to obtain this reference.

function result The result code `errKCNoDefaultKeychain` indicates that no default keychain was found. The result code `errKCItemNotFound` indicates that no matching password item was found. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling `KCFindInternetPasswordWithPath` again.

Discussion

The `KCFindInternetPasswordWithPath` function finds the first Internet password item which matches the attributes you provide, including path information. The buffer specified in the `passwordData` parameter must be large enough to hold the password data, otherwise `KCFindInternetPasswordWithPath` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling `KCFindInternetPasswordWithPath` again. `KCFindInternetPasswordWithPath` optionally returns a reference to the found item.

You can also call the function `kcfindinternetpasswordwithpath` to find the first Internet password item matching specified attributes. The difference between the two functions is that `kcfindinternetpasswordwithpath` takes a pointer to a C string instead of a Pascal string in the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes

Available beginning with Keychain Manager 2.0.

KCAddGenericPassword

Adds a new generic password to the default keychain.

```
OSStatus KCAddGenericPassword (
                    StringPtr serviceName,
                    StringPtr accountName,
                    UInt32 passwordLength,
                    const void *passwordData,
                    KCItemRef *item);
```

`serviceName`
: A pointer to a Pascal string containing an application-defined service name.

`accountName`
: A pointer to a Pascal string containing an application-defined account name.

`passwordLength`
: The length of the password data to be stored.

`passwordData`
: A pointer to a buffer which will hold the returned password data. Before calling `KCAddGenericPassword`, allocate enough memory for the buffer to hold the data you want to store.

`item`
: On return, a pointer to a reference to the added item. Pass `NULL` if you don't want to obtain this reference.

function result
: The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCDuplicateItem` indicates that you tried to add a password that already exists in the keychain. The result code `errKCDataTooLarge` indicates that you tried to add more data than is allowed for a record of this type.

Discussion

The `KCAddGenericPassword` function adds a new generic password to the default keychain. Required parameters to identify the password are `serviceName` and `accountName`, which are application-defined strings. `KCAddGenericPassword` optionally returns a reference to the newly added item.

You can use `KCAddGenericPassword` to add passwords for accounts other than Internet or Appleshare. For example, you might add passwords for your database or scheduling programs.

You can also call the function `kcaddgenericpassword` to add a new generic password to the default keychain. The difference between the two functions is that `kcaddgenericpassword` takes a pointer to a C string instead of a Pascal string in the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes
Available beginning with Keychain Manager 1.0.

KCFindGenericPassword

Finds the first generic password in the default keychain matching specified parameters.

```
OSStatus KCFindGenericPassword(
                    StringPtr serviceName,
                    StringPtr accountName,
                    UInt32 maxLength,
                    void *passwordData,
                    UInt32 *actualLength,
                    KCItemRef *item);
```

`serviceName`
A pointer to a Pascal string containing an application-defined service name. Pass `NULL` to match any service name.

`accountName`
A pointer to a Pascal string containing an application-defined account name. Pass `NULL` to match any account name.

`maxLength`
The length of the buffer pointed to by `passwordData`. Pass 0 if you want to obtain the item reference but not the password data. In this case, you must also pass `NULL` in the `passwordData` parameter.

`passwordData`
A pointer to a buffer which will hold the returned password data. Before calling `KCFindGenericPassword`, allocate enough memory for the buffer to hold the data you want to store. Pass `NULL` if you want to obtain the item reference but not the password data. In this case, you must also pass 0 in the `maxLength` parameter. On return, a pointer to the returned password data.

`actualLength`
: On return, the actual length of the password data that was retrieved. If the buffer pointed to by `passwordData` is smaller than the actual length of the data, `KCFindGenericPassword` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling `KCFindGenericPassword` again.

`item`
: On return, a pointer to a reference to the found item. Pass `NULL` if you don't want to obtain this reference.

function result The result code `errKCNoDefaultKeychain` indicates that no default keychain was found. The result code `errKCItemNotFound` indicates that no matching password item was found. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling `KCFindGenericPassword` again.

Discussion

The `KCFindGenericPassword` function finds the first generic password item which matches the attributes you provide. The buffer specified in the `passwordData` parameter must be large enough to hold the password data, otherwise `KCFindGenericPassword` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling `KCFindGenericPassword` again. `KCFindGenericPassword` optionally returns a reference to the found item.

You can also call the function `kcfindgenericpassword` to find the first generic password matching specified attributes. The difference between the two functions is that `kcfindgenericpassword` takes a pointer to a C string instead of a Pascal string in the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes

Available beginning with Keychain Manager 1.0.

Creating a Keychain Item Reference

`KCNewItem` — Creates a reference to a keychain item. (page 63)

KCReleaseItem — Disposes of the memory occupied by a keychain item reference. (page 64)

KCNewItem

Creates a reference to a keychain item.

```
OSStatus KCNewItem (
                    KCItemClass itemClass,
                    OSType itemCreator,
                    UInt32 length,
                    const void *data,
                    KCItemRef *item);
```

itemClass	The type of item (that is, a certificate, AppleShare password, Internet password, or generic password) that you want to create. See "Keychain Item Class Constants" (page 104) for a description of possible values.
itemCreator	The creator code of the application that owns this item.
length	The length of the data to be stored in this item.
data	A pointer to a buffer containing the data to be stored in this item. Before calling KCNewItem, allocate enough memory for the buffer to hold the data you want to store.
item	On return, a pointer to a reference to the newly-created item.
function result	The Memory Manager result code memFullErr indicates that you did not allocate enough memory in the current heap to create the item.

Discussion

The KCNewItem function creates a new keychain item from the specified parameters. Note that a copy of the data buffer pointed to by data is stored in the item.

Version Notes

Available beginning with Keychain Manager 1.0.

Special Considerations
If you want to store an item permanently, you must call the function `KCAddItem` (page 70) after calling `KCNewItem`. When the item reference is no longer required, call the function `KCReleaseItem` (page 64) to deallocate memory occupied by the item. You should not use the reference after it has been released.

KCReleaseItem

Disposes of the memory occupied by a keychain item reference.

```
OSStatus KCReleaseItem (KCItemRef *item);
```

`item`
: A pointer to a keychain item reference. Pass the keychain item reference whose memory you want to release. On return, the reference is set to `NULL` and should not be used again.

Discussion
You should call the `KCReleaseItem` function to release the memory occupied by a keychain item reference when you are finished with it.

Version Notes
Available beginning with Keychain Manager 1.0

Setting and Obtaining Keychain Item Attribute Data

`KCSetAttribute` — Sets or modifies keychain item attribute data specified by a pointer to an attribute structure. (page 65)

`KCGetAttribute` — Obtains keychain item attribute data specified by a pointer to an attribute structure. (page 66)

`KCSetData` — Sets or modifies keychain item attribute data specified by the data length and a pointer to data. (page 67)

`KCGetData` — Obtains keychain item attribute data specified by the data length and a pointer to data. (page 68)

KCSetAttribute

Sets or modifies keychain item attribute data specified by a pointer to an attribute structure.

```
OSStatus KCSetAttribute (
                    KCItemRef item,
                    KCAttribute *attr);
```

`item`
: A reference to the keychain item whose attribute data you wish to modify.

`attr`
: A pointer to a structure of t ype `KCAttribute` (page 86) containing keychain item attribute data. Before calling `KCSetAttribute`, fill in the `tag`, `length`, and `data` fields of this structure with the tag identifying the attribute you wish to modify or set, the length of the attribute data you wish to set, and a pointer to that data, respectively.

function result The result code `errKCInvalidItemRef` indicates that the keychain item reference was invalid. The result code `errKCNoSuchAttr` indicates that the item attribute you wish to set is undefined for the specified item. The result code `errKCDataTooLarge` indicates that more data was supplied than is allowed for this attribute.

Discussion

You can call the `KCSetAttribute` function to set or modify keychain item attribute data. You can also call the function `KCSetData` (page 67) to set or modify attribute data. The difference between the two functions is that `KCSetData` requires that you pass the length of the data and a pointer to that data as separate parameters rather than fields in a keychain item attribute structure.

Before calling KCSetAttribute to access keychain item data, you must call the function `KCUnlock` (page 44) to unlock the keychain. The keychain must allow read/write access for you to successfully modify or set its contents.

You can only set or modify standard item attributes identified by the tag constants `kDescriptionKCItemAttr`, `kCommentKCItemAttr`, `kLabelKCItemAttr`, `kCreatorKCItemAttr`, `kTypeKCItemAttr`, and `kCustomIconKCItemAttr`. In addition, each class of keychain item has attributes specific to that class which may be set or modified. See “Keychain Item Attribute Tag Constants” (page 99) for a description of item attribute tags.

Version Notes
Available beginning with Keychain Manager 1.0.

KCGetAttribute

Obtains keychain item attribute data specified by a pointer to an attribute structure.

```
OSStatus KCGetAttribute (
                    KCItemRef item,
                    KCAttribute *attr,
                    UInt32 *actualLength);
```

`item`
: A reference to the keychain item whose attribute data you wish to retrieve.

`attr`
: A pointer to a `KCAttribute` (page 86) structure containing keychain item attribute data. Before calling `KCGetAttribute`, fill in the `tag`, `length`, and `data` fields (the `data` field should contain a pointer to a buffer of sufficient length for the type of data to be returned). On return, `KCGetAttribute` fills in the `data` field with the retrieved attribute data.

`actualLength`
: On return, a pointer to the actual length of the attribute data. This may be more than the length you allocated in the length field of the attribute structure.

function result The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid. The result code `errKCNoSuchAttr` indicates that you tried to set an attribute which is undefined for this item class. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling `KCGetAttribute` again.

Discussion
You can call the `KCGetAttribute` function to obtain keychain item attribute data. You can also call the function `KCGetData` (page 68) to obtain attribute data. The difference between the two functions is that `KCGetData` requires that you pass the length of the data buffer as a separate parameter rather than as a field in a keychain item attribute structure. It passes back a pointer to the attribute data you requested and the actual length of that data, rather than filling in the fields of the attribute structure.

Before calling `KCGetAttribute` to access keychain item data, you must call the function `KCUnlock` (page 44) to unlock the keychain.

You can only obtain standard item attributes identified by the tag constants `kClassKCItemAttr`, `kCreationDateKCItemAttr`, `kModDateKCItemAttr`, `kDescriptionKCItemAttr`, `kCommentKCItemAttr`, `kLabelKCItemAttr`, `kCreatorKCItemAttr`, `kScriptCodeKCItemAttr`, and `kCustomIconKCItemAttr`. In addition, each class of keychain item has attributes specific to that class which may be obtained. See "Keychain Item Attribute Tag Constants" (page 99) for a description of item attribute tags.

Version Notes
Available beginning with Keychain Manager 1.0.

KCSetData

Sets or modifies keychain item attribute data specified by the data length and a pointer to data.

```
OSStatus KCSetData (
                    KCItemRef item,
                    UInt32 length,
                    const void *data);
```

`item`
: A reference to the keychain item whose data you wish to set.

`length`
: The length of the data buffer pointed to by the `data` parameter.

`data`
: A pointer to a buffer containing the data to be stored in this item. Before calling `KCSetData`, allocate enough memory for the buffer to hold the data you want to store.

function result The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid. The result code `errKCDataTooLarge` indicates that the data was too large for the supplied buffer. The result code `errKCDataNotModifiable` indicates that the data cannot be set for this item.

Discussion

You can call the KCSetData function to set or modify keychain item attribute data. You can also call the function KCSetAttribute (page 65) to set or modify attribute data. The difference between the two functions is that KCSetData requires that you pass the length of the data and a pointer to that data as separate parameters rather than fields in a keychain item attribute structure.

Before calling KCSetData to access keychain item data, you must call the function KCUnlock (page 44) to unlock the keychain. The keychain must allow read/write access for you to successfully modify or set its contents.

You can only set or modify standard item attributes identified by the tag constants kDescriptionKCItemAttr, kCommentKCItemAttr, kLabelKCItemAttr, kCreatorKCItemAttr, kTypeKCItemAttr, and kCustomIconKCItemAttr. In addition, each class of keychain item has attributes specific to that class which may be set or modified. See "Keychain Item Attribute Tag Constants" (page 99) for a description of item attribute tags.

Version Notes

Available beginning with Keychain Manager 1.0.

KCGetData

Obtains keychain item attribute data specified by the data length and a pointer to data.

```
OSStatus KCGetData (
                    KCItemRef item,
                    UInt32 maxLength,
                    void *data
                    UInt32 *actualLength);
```

item

A reference to the keychain item whose data you wish to retrieve.

maxLength

The length of the data buffer pointed to by the data parameter.

data

A pointer to a buffer which will hold the returned data. Before calling KCGetData, allocate enough memory for the buffer to hold the data you want to store. On return, a pointer to the attribute data you requested.

`actualLength`
On return, a pointer to the actual length of the data being retrieved. If the buffer pointed to by `data` is smaller than the actual length of the data, `KCGetData` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling `KCGetData` again.

function result The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling `KCGetData` again. The result code `errKCDataNotModifiable` indicates that the data is not available for this item.

Discussion

You can call the `KCGetData` function to obtain keychain item attribute data. You can also call the function `KCGetAttribute` (page 66) to obtain attribute data. The difference between the two functions is that `KCGetAttribute` requires that you pass the length of the data buffer as a field in a keychain item attribute structure rather than as a separate parameter. It passes a pointer to the attribute structure rather than a pointer to the attribute data and the actual length of that data as separate parameters.

Before calling `KCGetData` to access keychain item data, you must call the function `KCUnlock` (page 44) to unlock the keychain. You cannot call `KCGetData` for a private key. `KCGetData` requires that you unlock the keychain in order to access item data.

You can only obtain standard item attributes identified by the tag constants `kClassKCItemAttr`, `kCreationDateKCItemAttr`, `kModDateKCItemAttr`, `kDescriptionKCItemAttr`, `kCommentKCItemAttr`, `kLabelKCItemAttr`, `kCreatorKCItemAttr`, `kScriptCodeKCItemAttr`, and `kCustomIconKCItemAttr`. In addition, each class of keychain item has attributes specific to that class which may be obtained. See “Keychain Item Attribute Tag Constants” (page 99) for a description of item attribute tags.

Version Notes

Available beginning with Keychain Manager 1.0.

Manipulating Keychain Items

`KCAddItem` — Adds a keychain item to the default keychain. (page 70)

`KCDeleteItem` — Deletes a keychain item from the default keychain. (page 70)

`KCUpdateItem` — Updates a keychain item. (page 71)

`KCCopyItem` — Copies a keychain item from one keychain to another. (page 72)

KCAddItem

Adds a keychain item to the default keychain.

```
OSStatus KCAddItem (KCItemRef item);
```

`item`
: A reference to the keychain item you wish to add. If you pass an existing item in the keychain, the item is updated. If you pass an item that has not been previously added to the keychain and an identical item already exists in the keychain, `KCAddItem` returns the result code `errKCDuplicateItem`.

function result
: The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid. The result code `errKCDuplicateItem` indicates that you tried to add a new item that already exists in the keychain.

Discussion

You can use the `KCAddItem` function to add a keychain item to the permanent data store of the default keychain. If you want to add an item to a specific keychain, bracket this call with the functions `KCGetDefaultKeychain` (page 41) and `KCSetDefaultKeychain` (page 41). Calling these functions enable you to change where items are added. KCAddItem may display the Unlock Keychain dialog if the keychain containing the item is currently locked.

Version Notes

Available beginning with Keychain Manager 1.0.

KCDeleteItem

Deletes a keychain item from the default keychain.

```
OSStatus KCDeleteItem (KCItemRef item);
```

`item`
: A reference to the keychain item you wish to delete. If you pass an item that has not been previously added to the keychain, `KCDeleteItem` does nothing and returns `noErr`.

function result The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid.

Discussion
You can use the `KCDeleteItem` function to delete a keychain item from the permanent data store of the default keychain. `KCDeleteItem` may display the Unlock Keychain dialog if the keychain containing the item is currently locked.

Version Notes
Available beginning with Keychain Manager 1.0.

Special Considerations
`KCDeleteItem` does not dispose the memory occupied by the item reference. To do so, call the function `KCReleaseItem` (page 64) when you are finished with an item.

KCUpdateItem

Updates a keychain item.

```
OSStatus KCUpdateItem (KCItemRef item);
```

`item`
: A reference to the keychain item you wish to update. If you pass an item that has not been previously added to the keychain, `KCUpdateItem` does nothing and returns `noErr`.

function result The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid.

Discussion
You can use the `KCUpdateItem` function to update a keychain item after changing its attributes or data. The item is written to the keychain's permanent data store. `KCUpdateItem` may display the Unlock Keychain dialog if the keychain containing the item is currently locked.

Version Notes
Available beginning with Keychain Manager 1.0.

KCCopyItem

Copies a keychain item from one keychain to another.

```
OSStatus KCCopyItem (
                    KCItemRef item,
                    KCRef destKeychain,
                    KCItemRef *copy);
```

`item`	A reference to the keychain item you wish to copy.
`destKeychain`	A reference to the keychain into which the item is to be copied.
`copy`	A pointer to a reference to the new copied keychain item.
function result	The result code `errKCReadOnly` indicates that the destination keychain is read only. The result code `errKCNoSuchClass` indicates that the item has an invalid keychain item class. The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid.

Discussion
You can use the `KCCopyItem` function to copy a keychain item from one keychain to another. `KCCopyItem` returns the copied item in the `copy` parameter. `KCCopyItem` may display the Unlock Keychain dialog if the keychain containing the item is currently locked.

Version Notes
Available beginning with Keychain Manager 2.0.

KCGetKeychain

Retrieves the location of a keychain item.

```
OSStatus KCGetKeychain(
                    KCItemRef item,
                    KCRef *keychain);
```

`item`
: A reference to the keychain item whose keychain location you wish to determine. If you pass a reference to a keychain item whose keychain is locked, `KCGetKeychain` returns the result code `errKCInvalidItemRef`.

`keychain`
: On return, a pointer to the keychain containing the specified item.

function result The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid.

Discussion

You can use the `KCGetKeychain`function to find the location of keychain items. The search is only performed on unlocked keychains. `KCDeleteItem` may display the Unlock Keychain dialog if the keychain containing the item is currently locked.

Version Notes

Available beginning with Keychain Manager 2.0.

Special Considerations

The keychain reference returned by `KCGetKeychain` should be released by calling the function `KCReleaseItem` (page 64).

Searching for Keychain Items

`KCFindFirstItem` — Finds the first keychain item in a specified keychain that matches specified attributes. (page 74)

`KCFindNextItem` — Finds the next keychain item matching the previously specified search criteria. (page 75)

`KCReleaseSearch` — Disposes of the memory occupied by a search criteria reference. (page 76)

KCFindFirstItem

Finds the first keychain item in a specified keychain that matches specified attributes.

```
OSStatus KCFindFirstItem (
                    KCRef keychain
                    const KCAttributeList *attrList,
                    KCSearchRef *searchRef,
                    KCItemRef *item);
```

`keychain`
: A reference to the keychain that you wish to search. If you pass a locked keychain, the Unlock Keychain dialog is displayed. If you pass `NULL`, `KCFindFirstItem` search all unlocked keychains.

`attrList`
: A pointer to a list of 0 or more structures containing information about the keychain item attributes to be matched. Pass `NULL` to match any attribute.

`searchRef`
: On return, a pointer to a reference to the current search criteria.

`item`
: On return, a pointer to the first matching keychain item.

function result The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCItemNotFound` indicates that no matching keychain item was found. The result code `errKCNoSuchAttr` indicates that the specified attribute is undefined for this item class.

Discussion

The `KCFindFirstItem` function finds the first keychain item matching a list of zero or more specified attributes in the specified keychain. `KCFindFirstItem` returns a reference to the matching item and the current search criteria. You can use the returned search criteria for subsequent calls to the function `KCFindNextItem` (page 75).

Version Notes

Available beginning with Keychain Manager 1.0.

Special Considerations

When you are completely finished with the search, call the functions `KCReleaseItem` (page 64) and `KCReleaseSearch` (page 76) to release the keychain item reference and search criteria reference, respectively.

KCFindNextItem

Finds the next keychain item matching the previously specified search criteria.

```
OSStatus KCFindNextItem (
                    KCSearchRef searchRef,
                    KCItemRef *item);
```

`searchRef`
: A reference to the previously-specified search criteria. Pass the reference passed back in the `searchRef` parameter of the function <codeXRefText>KCFindFirstItem (page 74).

`item`
: On return, a pointer to the next matching keychain item, if any.

function result The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCItemNotFound` indicates that no matching keychain item was found. The result code `errKCInvalidSearchRef` indicates that the specified search reference was invalid.

Discussion

The `KCFindNextItem` function finds the next keychain item matching the search criteria previously specified by a call to `KCFindFirstItem` (page 74). `KCFindNextItem` returns a reference to the matching item, if any.

Version Notes

Available beginning with Keychain Manager 1.0.

Special Considerations

When you are completely finished with the search, call the functions `KCReleaseItem` (page 64) and `KCReleaseSearch` (page 76) to release the keychain item reference and search criteria reference, respectively.

KCReleaseSearch

Disposes of the memory occupied by a search criteria reference.

```
OSStatus KCReleaseSearch (KCSearchRef *search);
```

`search`	A pointer to a search criteria reference. Pass the search criteria reference whose memory you want to release. On return, the reference is set to `NULL` and should not be used again.
function result	The result code `errKCInvalidSearchRef` indicates that the specified search reference was invalid.

Discussion

You should call the `KCReleaseSearch` function to release the memory occupied by a search criteria reference when you are completely finished with a search performed by calling the functions `KCFindFirstItem` (page 74) or `KCFindNextItem` (page 75).

Version Notes

Available beginning with Keychain Manager 1.0

Working With Certificates

`KCFindX509Certificates` — Finds certificates in a specified keychain that match the specified search criteria. (page 76)

`KCChooseCertificate` — Displays a list of certificates that the user can choose from. (page 77)

KCFindX509Certificates

Finds certificates in a specified keychain that match the specified search criteria.

```
OSStatus KCFindX509Certificates (
                    KCRef keychain,
                    CFStringRef name,
                    CFStringRef emailAddress,
                    KCCertSearchOptions options,
                    CFMutableArrayRef *certificateItems);
```

`keychain`
: A reference to the keychain you want to search. If the specified keychain is locked, the Unlock Keychain dialog is displayed.

`name`
: A pointer to a C string containing the certificate owner's common name.

`emailAddress`
: A pointer to a C string containing the certificate owner's e-mail address.

`options`
: The search criteria used when retrieving certificates. You can use the masks described in "Certificate Search Option Mask Constants" (page 92) to set the criteria.

`certificateItems`
: On return, a pointer to an array reference of the certificate items that were found. Pass `NULL` if you don't want to obtain these references.

function result The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCBufferTooSmall` indicates that the certificate data was too large for the supplied buffer. In this case, your application must allocate a new buffer of sufficient size before calling `KCFindX509Certificates` again. The result code `errKCItemNotFound` indicates that no matching certificate object was found.

Version Notes

Available beginning with Keychain 2.0.

Carbon Porting Notes

The `KCFindX509Certificates` function is fully supported in Mac OS 9. In the first release of Mac OS X, `KCFindX509Certificates` returns `unImpErr`. Your application should handle this error accordingly.

KCChooseCertificate

Displays a list of certificates that the user can choose from.

```
OSStatus KCChooseCertificate (
                    CFArrayRef items,
                    KCItemRef *certificate,
                    CFArrayRef policyOIDs,
                    KCVerifyStopOn stopOn);
```

`items`
: A Core Foundation array of certificate keychain item references.

`certificate`
: On return, a pointer to the certificate keychain item. This is returned if the `items` array contains one keychain item. In this case, no user interface is displayed.

`policyOIDs`
: A Core Foundation array of policy OIDs that determine the trust policy. To obtain a pointer to an array of policy OIDs for Macintosh file signing, call the function `SecMacGetDefaultPolicyOIDs`.

`stopOn`
: One of the constants defined by the `KCVerifyStopOn` enumeration. For a description of these values, see "Verification Criteria Constants" (page 107).

function result The result code `userCanceledErr` indicates that the user cancelled out from the user interface presented.

Discussion

The `KCChooseCertificate` function displays a list of the certificates the user can chose if the `items` array contains at least two keychain items. Otherwise, it returns the single keychain item in the `certificate` parameter with no user interface.

Version Notes

Available beginning with Keychain 2.0.

Carbon Porting Notes

The `KCChooseCertificate` function is fully supported in Mac OS 9. In the first release of Mac OS X, `KCChooseCertificate` returns `unImpErr`. Your application should handle this error accordingly.

Managing User Interaction

`KCSetInteractionAllowed` — Tells Keychain functions that display user interface whether to do so. (page 79)

`KCIsInteractionAllowed` — Reports whether user interaction is allowed. (page 79)

KCSetInteractionAllowed

Tells Keychain functions that display user interface whether to do so.

```
OSStatus KCSetInteractionAllowed (Boolean state);
```

`state`	A flag that toggles the user interface state. If you pass `true`, user interaction is allowed. By default, user interaction is allowed. If you pass `false`, user interaction is not allowed. In this case, Keychain functions that normally display a user interface will instead return an error.

Discussion

Failing to re-enable user interaction will affect other clients of the Keychain Manager. The interaction allowed state is reset when the machine reboots to the default state, that is, user interaction allowed.

Version Notes

Available beginning with Keychain Manager 2.0.

KCIsInteractionAllowed

Reports whether user interaction is allowed.

```
Boolean KCIsInteractionAllowed (void);
```

function result	A `Boolean` value indicating whether user interaction is permitted. If `true`, user interaction is allowed, and Keychain Manager is free to show a user interface when needed.

Version Notes

Available beginning with Keychain Manager 2.0.

Registering Your Keychain Event Callback Function

`KCAddCallback` — Registers your keychain event callback function. (page 80)

`KCRemoveCallback` — Unregisters your keychain event callback function. (page 81)

KCAddCallback

Registers your keychain event callback function.

```
OSStatus KCAddCallback  (
                KCCallbackUPP callbackProc,
                KCEventMask eventMask,
                void *userContext);
```

`callbackProc`
: A Universal Procedure Pointer (UPP) to your keychain event callback function, described in `KCCallbackProcPtr` (page 84). You indicate the type of keychain events you want to receive by passing a bitmask of the desired events in the `eventMask` parameter. To create a UPP to your callback function, call the function `NewKCCallbackUPP` (page 82).

`eventMask`
: A bitmask indicating the keychain events that your application wishes to be notified of. See "Keychain Event Mask Constants" (page 97) for a description of this bitmask. The Keychain Manager tests this mask to determine the keychain events that you wish to receive, and passes these events in the `keychainEvent` parameter of your callback function. See "Keychain Event Constants" (page 96) for a description of these events.

`userContext`
: A pointer to application-defined storage that will be passed to your callback function. Your application can use this to associate any particular call of `KCAddCallback` with any particular call of your keychain event callback function.

function result The result code `errKCDuplicateCallback` indicates that your callback function is already registered.

Discussion
You can register your callback function by passing a UPP to it in the `callbackProc` parameter of the `KCAddCallback` function. Once you done so, the Keychain Manager calls the function `InvokeKCCallbackUPP` (page 82) when the keychain event specified in the `eventMask` parameter occurs. In turn, `InvokeKCCallbackUPP` passes the keychain event, information about the event, and application-defined storage to your keychain event callback function.

Version Notes
Available beginning with Keychain 1.0.

KCRemoveCallback

Unregisters your keychain event callback function.

```
OSStatus KCRemoveCallback (KCCallbackUPP callbackProc);
```

`callbackProc`
: A Universal Procedure Pointer (UPP) to your keychain event callback function that was previously registered with the function `KCAddCallback` (page 80).

function result
: The result code `errKCInvalidCallback` indicates that the callback function was not previously registered.

Discussion

After you pass a UPP to your keychain event callback function to the `KCRemoveCallback` function, it will no longer be called by the Keychain Manager.

Version Notes

Available beginning with Keychain 1.0.

Special Considerations

After calling `KCRemoveCallback`, you should call the function `DisposeKCCallbackUPP` (page 83) to dispose of the UPP to your callback function.

Creating and Managing Universal Procedure Pointers

You can use these functions to create and manage universal procedure pointers (UPPs) to your keychain event callback function.

`NewKCCallbackUPP` — Creates a UPP to your keychain event callback function. (page 82)

`InvokeKCCallbackUPP` — Invokes your keychain event callback function. (page 82)

`DisposeKCCallbackUPP` — Disposes of a UPP to your keychain event callback function. (page 83)

NewKCCallbackUPP

Creates a UPP to your keychain event callback function.

```
KCCallbackUPP NewKCCallbackUPP (KCCallbackProcPtr userroutine);
```

`userroutine`
: A pointer to your keychain event callback function. For information on how to create a keychain event callback, see `KCCallbackProcPtr` (page 84).

function result A UPP to your callback function. You can register your callback function by passing this UPP in the `callbackProc` parameter of the function `KCAddCallback` (page 80).

Discussion

The `NewKCCallbackUPP` function creates a pointer to your keychain event callback function. You pass a pointer to your callback function in the `callbackProc` parameter of the function `KCAddCallback` (page 80) if you want your application to receive data transfer events.

Version Notes

Available beginning with Keychain 1.0.

Special Considerations

When you are finished with a UPP to your keychain event callback function, you should dispose of it by calling the function `DisposeKCCallbackUPP` (page 83).

InvokeKCCallbackUPP

Invokes your keychain event callback function.

```
OSStatus InvokeKCCallbackUPP(
                    KCEvent keychainEvent,
                    KCCallbackInfo *info,
                    void *userContext,
                    KCCallbackUPP userUPP);
```

`keychainEvent`
: The keychain events you want your application to receive. See "Keychain Event Constants" (page 96) for a description of possible values. The Keychain Manager tests the bitmask you pass in the `eventMask` parameter of the function `KCAddCallback` (page 80) to

determine which events to pass to your callback function. See "Keychain Event Mask Constants" (page 97) for a description of this bitmask.

`info`
: A pointer to a structure of type `KCCallbackInfo` (page 88) that provides information about the keychain event to your callback function. The Keychain Manager passes a pointer to this structure in the `info` parameter of your callback function.

`userContext`
: A pointer to application-defined storage. The Keychain Manager passes this value in the `userContext` parameter of your callback function. Your application can use this to associate any particular call of `InvokeKCCallbackUPP` with any particular call of the keychain event callback function.

`userUPP`
: A Universal Procedure Pointer to your keychain event callback function. For information on how to create a keychain event callback function, see `KCCallbackProcPtr` (page 84).

Discussion

The Keychain Manager calls the `InvokeKCCallbackUPP` function when you pass a UPP to your callback function in the `callbackProc` parameter of the function `KCAddCallback` (page 80), and the keychain event that you specified in the `eventMask` parameter occurs.

DisposeKCCallbackUPP

Disposes of a UPP to your keychain event callback function.

```
OSStatus DisposeKCCallbackUPP (KCCallbackUPP userUPP);
```

`userUPP`
: A Universal Procedure Pointer (UPP) to your keychain event callback function.

Discussion

When you are finished with a UPP to your keychain event callback function, you should dispose of it by calling the `DisposeKCCallbackUPP` function.

Version Notes

Available beginning with Keychain 1.0.

Keychain Manager Callback

`KCCallbackProcPtr` — Defines a pointer to your keychain event callback function. Your keychain event callback function handles events that occur when the user accesses the keychain. (page 84)

KCCallbackProcPtr

Defines a pointer to your keychain event callback function. Your keychain event callback function handles events that occur when the user accesses the keychain.

```
OSStatus *KCCallbackProcPtr
       (KCEvent keychainEvent, KCCallbackInfo *info, void *userContext);
```

You would declare your Keychain event callback function like this if you were to name it `MyKCCallback`:

```
OSStatus MyKCCallback(
                    KCEvent keychainEvent
                    KCCallbackInfo *info,
                    void *userContext);
```

`keychainEvent`

The keychain event that your application wishes to be notified of. See "Keychain Event Constants" (page 96) for a description of possible values. The type of event that can trigger your callback depends on the bitmask you passed in the `eventMask` parameter of the function `KCAddCallback` (page 80). For more information, see the discussion.

`info`

A pointer to a structure of type `KCCallbackInfo` (page 88). On return, the structure contains information about the keychain event that occurred. The Keychain Manager passes this information to your callback function via the `info` parameter of the function `InvokeKCCallbackUPP` (page 82).

`userContext`
: A pointer to application-defined storage that your application previously passed to the function `KCAddCallback` (page 80). You can use this value to perform operations like track which instance of a function is operating.

function result
: Your keychain event callback function should process the keychain event and return `noErr`.

Discussion

Your keychain event callback function handles those keychain events that you indicate. In order to be notified of these events, you must pass a UPP to your notification callback function in the `callbackProc` parameter of `KCAddCallback` (page 80). You indicate the type of data transfer events you want to receive via a bitmask in the `eventMask` parameter. When you no longer wish to receive notification of keychain events, you should call the function `KCRemoveCallback` (page 81) to dispose of the UPP to your keychain event callback function.

Keychain Manager Data Types

`AFPServerSignature` — Represents a 16-byte Apple File Protocol server signature block. (page 86)

`KCAttribute` — Contains information about a keychain item attribute. (page 86)

`KCAttributeList` — Lists attributes in a keychain item. (page 87)

`KCAttrType` — Identifies a keychain item attribute value. (page 87)

`KCCallbackInfo` — Contains information about a keychain event. (page 88)

`KCItemRef` — Represents a reference to a keychain item. (page 89)

`KCPublicKeyHash` — Represents a 20-byte public key hash. (page 89)

`KCRef` — Represents a reference to a keychain. (page 89)

`KCSearchRef` — Represents a reference to the current search criteria. (page 90)

AFPServerSignature

Represents a 16-byte Apple File Protocol server signature block.

```
typedef UInt8 AFPServerSignature[16];
```

Discussion

The `AFPServerSignature` type represents a 16-byte Apple File Protocol server signature block. You can pass a value of this type in the serverSignature parameter of the functions `KCAddAppleSharePassword` (page 48) and `KCFindAppleSharePassword` (page 49) to represent an Apple File Protocol server signature. You can use a value of this type with the keychain item attribute constant `kSignatureKCItemAttr` to specify an Apple File Protocol server signature.

KCAttribute

Contains information about a keychain item attribute.

```
struct KCAttribute {
       KCAttrType tag;
       UInt32 length;
       void *data;
   };
   typedef struct KCAttribute KCAttribute,
   typedef KCAttribute * KCAttributePtr;
```

Field descriptions

`tag`

Identifies a keychain item attribute value. See "Keychain Item Attribute Tag Constants" (page 99) for a description of the Apple-defined tag constants and the data types of the values they identify. Your application can create application-defined tags of type `KCAttrType` (page 87).

`length`

The length of the attribute data.

`data`

A pointer to the attribute data. When calling the function `KCSetAttribute` (page 65), you should set this field to a pointer to the attribute data you wish to add. When calling the function `KCGetAttribute` (page 66), you should set this field to a pointer to a buffer of sufficient length for the type of data to be returned. On return, this field contains the requested attribute data.

Discussion
The `KCAttribute` type represents a structure containing information about the attribute of a keychain item. It contains a tag that identifies a particular keychain item attribute value, the length of the attribute value, and a pointer to the attribute value. You can modify attribute data for a keychain item attribute by passing a pointer to this structure in the `attr` parameter of the function `KCSetAttribute` (page 65). The function `KCGetAttribute` (page 66) passes back a pointer to this structure in the `attr` parameter.

KCAttributeList

Lists attributes in a keychain item.

```
struct KCAttributeList {
       UInt32 count;
       KCAttribute *attr;
   };
   typedef struct KCAttributeList KCAttributeList;
```

Field descriptions

`count`	The number of keychain item attribute structures in this list.
`attr`	A pointer to the first keychain item attribute structure in this list.

Discussion
The `KCAttributeList` type represents a list of structures containing information about the attributes in a keychain item. You pass a pointer to this list of 0 or more structures in the `attrList` parameter of the function `KCFindFirstItem` (page 74) to indicate the attributes to be matched.

KCAttrType

Identifies a keychain item attribute value.

```
typedef OSType KCAttrType;
```

Discussion
The `KCAttrType` type represents a tag that identifies a keychain item attribute value. You can use this value in the `tag` field of the structure `KCAttribute` (page 86) to identify the keychain item attribute value you wish to set or obtain. See “Keychain

Item Attribute Tag Constants" (page 99) for a description of the Apple-defined tag constants and the data types of the values they identify. Your application can create application-defined tags of type `KCAttrType`.

KCCallbackInfo

Contains information about a keychain event.

```
struct KCCallbackInfo{
   UInt32 version;
   KCItemRef item;
   ProcessSerialNumber processID;
   EventRecord event;
   KCRef keychain;
   };
   typedef struct KCCallbackInfo KCCallbackInfo;
```

Field descriptions

`version`
: The version of this structure.

`item`
: A reference to the keychain item in which the event occurred. If the event did not involve an item, this field is not valid.

`processID`
: A 64-bit quantity containing the process serial number of the process in which the event occurred.

`event`
: The keychain event that occurred. If the event is a system event as indicated by `kSystemKCEvent`, the Keychain client can process events. If the event is not a system event, this field is not valid.

`keychain`
: A reference to the keychain in which the event occurred. If the event did not involve a keychain, this field is not valid.

Discussion

The `KCCallbackInfo` type represents a structure that contains information about the keychain event that your application wants to be notified of. The Keychain Manager passes a pointer to this structure in the `info` parameter of your callback function via the function `InvokeKCCallbackUPP` (page 82), which invokes your callback function. For information on how to write a keychain event callback function, see `KCCallbackProcPtr` (page 84).

KCItemRef

Represents a reference to a keychain item.

```
typedef struct OpaqueKCItemRef* KCItemRef;
```

Discussion

The `KCItemRef` type represents a reference to an opaque structure that identifies a keychain item. You should call the function `KCNewItem` (page 63) to create a keychain item reference. The function `KCReleaseItem` (page 64) disposes of a keychain item reference when no longer needed. You pass a reference of this type to Keychain Manager functions that operate on a keychain item in some way.

KCPublicKeyHash

Represents a 20-byte public key hash.

```
typedef UInt8 KCPublicKeyHash[20];
```

Discussion

The `KCPublicKeyHash` type represents a hash of a public key. You can use the `tag constant kPublicKeyHashKCItemAttr`, described in "Keychain Item Attribute Tag Constants" (page 99), to set or retrieve a certificate attribute value of this type.

Carbon Porting Notes

The `KCPublicKeyHash` type is fully supported in Mac OS 9 but will not be supported in the first release of OS X.

KCRef

Represents a reference to a keychain.

```
typedef struct OpaqueKCRef* KCRef;
```

Discussion

The `KCRef` type represents a reference to an opaque structure that identifies a keychain. You should call the function `KCMakeKCRefFromFSSpec` (page 37) or `KCMakeKCRefFromAlias` (page 38) to create a keychain reference. The function `KCReleaseKeychain` (page 39) disposes of a keychain reference when no longer needed. You pass a reference of this type to Keychain Manager functions that operate on a keychain in some way.

`KCSearchRef`

Represents a reference to the current search criteria.

```
typedef struct OpaqueKCSearchRef* KCSearchRef;
```

Discussion

The `KCSearchRef` type represents a reference to an opaque structure that identifies the current search criteria. The function `KCFindFirstItem` (page 74) passes back a reference of this type in the `search` parameter for subsequent calls to the function `KCFindNextItem` (page 75). You must release this reference when you are finished with a search by calling the function `KCReleaseSearch` (page 76).

Keychain Manager Constants

`Authentication Type Constants` — Represent the type of authentication to use in storing and retrieving Internet passwords. (page 91)

`Certificate Search Option Mask Constants` — Represent a mask that specifies the search criteria to use when retrieving certificates. (page 92)

`Default Authentication Type Constant` — Indicates that any authentication type can be used. (page 94)

`Default Port Constant` — Indicates that any port can be used. (page 95)

`Default Protocol Constant` — Indicates that any protocol can be used. (page 95)

`Keychain Event Constants` — Identify Keychain-related events. (page 96)

`Keychain Event Mask Constants` — Represent a mask that indicates the Keychain-related events your notification callback function will receive. (page 97)

`Keychain Item Attribute Tag Constants` — Represent tags that identify keychain item attribute values. (page 99)

`Keychain Item Class Constants` — Identify the type of the keychain item you want to create. (page 104)

`Keychain Protocol Type Constants` — Represent the type of protocol to use in storing and retrieving Internet passwords. (page 105)

`Keychain Status Mask Constants` — Represent a mask identifying the status of a keychain. (page 106)

`Verification Criteria Constants` — Represent the verification criteria to use in selecting certificates. (page 107)

Authentication Type Constants

Represent the type of authentication to use in storing and retrieving Internet passwords.

```
enum {
   kKCAuthTypeNTLM          = 'ntlm',
   kKCAuthTypeMSN           = 'msna',
   kKCAuthTypeDPA           = 'dpaa',
   kKCAuthTypeRPA           = 'rpaa',
   kKCAuthTypeHTTPDigest    = 'httd',
   kKCAuthTypeDefault       = 'dflt'
   };
   typedef FourCharCode KCAuthType;
```

Constant descriptions

`kKCAuthTypeNTLM`
Specifies Windows NT LAN Manager authentication.

`kKCAuthTypeMSN`
Specifies Microsoft Network authentication.

`kKCAuthTypeDPA`
Specifies Distributed Password authentication.

`kKCAuthTypeRPA`
Specifies Remote Password authentication.

`kKCAuthTypeHTTPDigest`
Specifies HTTP Digest Access authentication.

`kKCAuthTypeDefault`
Specifies default authentication.

Discussion

The `KCAuthType` enumeration defines constants you can use to identify the type of authentication to use in storing and retrieving Internet passwords. You can pass a constant of this type in the `authType` parameter of the functions `KCAddInternetPassword` (page 52), `KCAddInternetPasswordWithPath` (page 53), `KCFindInternetPassword` (page 55), and `KCFindInternetPasswordWithPath` (page 57).

Certificate Search Option Mask Constants

Represent a mask that specifies the search criteria to use when retrieving certificates.

```
enum {
   kCertSearchShift             = 0, /* start at bit 0 */
   kCertSearchSigningIgnored    = 0,
   kCertSearchSigningAllowed    = 1 << (kCertSearchShift + 0),
   kCertSearchSigningDisallowed = 1 << (kCertSearchShift + 1),
   kCertSearchSigningMask       = ((kCertSearchSigningAllowed) |
                                    (kCertSearchSigningDisallowed)),
   kCertSearchVerifyIgnored     = 0,
   kCertSearchVerifyAllowed     = 1 << (kCertSearchShift + 2),
   kCertSearchVerifyDisallowed  = 1 << (kCertSearchShift + 3),
   kCertSearchVerifyMask        = ((kCertSearchVerifyAllowed) |
                                    (kCertSearchVerifyDisallowed)),
   kCertSearchEncryptIgnored    = 0,
   kCertSearchEncryptAllowed    = 1 << (kCertSearchShift + 4),
   kCertSearchEncryptDisallowed = 1 << (kCertSearchShift + 5),
   kCertSearchEncryptMask       = ((kCertSearchEncryptAllowed) |
                                    (kCertSearchEncryptDisallowed)),
   kCertSearchDecryptIgnored    = 0,
   kCertSearchDecryptAllowed    = 1 << (kCertSearchShift + 6),
   kCertSearchDecryptDisallowed = 1 << (kCertSearchShift + 7),
   kCertSearchDecryptMask       = ((kCertSearchDecryptAllowed) |
                                    (kCertSearchDecryptDisallowed)),
   kCertSearchWrapIgnored       = 0,
   kCertSearchWrapAllowed       = 1 << (kCertSearchShift + 8),
   kCertSearchWrapDisallowed    = 1 << (kCertSearchShift + 9),
   kCertSearchWrapMask          = ((kCertSearchWrapAllowed) |
                                    (kCertSearchWrapDisallowed)),
   kCertSearchUnwrapIgnored     = 0,
   kCertSearchUnwrapAllowed     = 1 << (kCertSearchShift + 10),
   kCertSearchUnwrapDisallowed  = 1 << (kCertSearchShift + 11),
   kCertSearchUnwrapMask        = ((kCertSearchUnwrapAllowed) |
                                    (kCertSearchUnwrapDisallowed)),
   kCertSearchPrivKeyRequired   = 1 << (kCertSearchShift + 12),
   kCertSearchAny               = 0
   };
   typedef UInt32 KCCertSearchOptions;
```

Constant descriptions

`kCertSearchShift`

`kCertSearchSigningIgnored`

`kCertSearchSigningAllowed`

`kCertSearchSigningDisallowed`

`kCertSearchSigningMask`

`kCertSearchVerifyIgnored`

`kCertSearchVerifyAllowed`

`kCertSearchVerifyDisallowed`

`kCertSearchVerifyMask`

`kCertSearchEncryptIgnored`

`kCertSearchEncryptAllowed`

`kCertSearchEncryptDisallowed`

`kCertSearchEncryptMask`

`kCertSearchDecryptIgnored`

`kCertSearchDecryptAllowed`

`kCertSearchDecryptDisallowed`

`kCertSearchDecryptMask`

`kCertSearchWrapIgnored`

`kCertSearchWrapAllowed`

`kCertSearchWrapDisallowed`

`kCertSearchWrapMask`

`kCertSearchUnwrapIgnored`

`kCertSearchUnwrapAllowed`

`kCertSearchUnwrapDisallowed`

`kCertSearchUnwrapMask`

`kCertSearchPrivKeyRequired`

`kCertSearchAny`

Discussion

The `KCCertSearchOptions` enumeration defines masks that you can use to set the search criteria to use when retrieving certificates. in the `options` parameter of the function `KCFindX509Certificates` (page 76).

Carbon Porting Notes

The `KCCertSearchOptions` enumeration is fully supported in Mac OS 9 but will not be supported in the first release of Mac OS X.

Default Authentication Type Constant

Indicates that any authentication type can be used.

```
 const OSType kAnyAuthType = 0L;
```

Constant descriptions

`kAnyAuthType`

Specifies that any authentication type can be used.

Discussion

You can pass the `kAnyAuthType` constant in the `authType` parameter of the functions `KCAddInternetPassword` (page 52), `KCAddInternetPasswordWithPath` (page 53), `KCFindInternetPassword` (page 55), and `KCFindInternetPasswordWithPath` (page 57) to indicate that any authentication scheme can be used during the add or search operation.

Default Port Constant

Indicates that any port can be used.

```
const UInt16 kAnyPort           = 0;
```

Constant descriptions

`kAnyPort`
: Specifies that any port can be used.

Discussion

You can pass the `kAnyPort` constant in the `port` parameter of the functions `KCAddInternetPassword` (page 52), `KCAddInternetPasswordWithPath` (page 53), `KCFindInternetPassword` (page 55), and `KCFindInternetPasswordWithPath` (page 57) to indicate that any port can be used during the add or search operation.

Default Protocol Constant

Indicates that any protocol can be used.

```
const OSType kAnyProtocol = 0L
```

Constant descriptions

`kAnyProtocol`
: Specifies that any protocol can be used.

Discussion

You can pass the `kAnyProtocol` constant in the `protocol` parameter of the functions `KCAddInternetPassword` (page 52), `KCAddInternetPasswordWithPath` (page 53), `KCFindInternetPassword` (page 55), and `KCFindInternetPasswordWithPath` (page 57) to indicate that any protocol can be used during the add or search operation.

Keychain Event Constants

Identify Keychain-related events.

```
enum {
       kIdleKCEvent             = 0,
       kLockKCEvent             = 1,
       kUnlockKCEvent           = 2,
       kAddKCEvent              = 3,
       kDeleteKCEvent           = 4,
       kUpdateKCEvent           = 5,
       kChangeIdentityKCEvent   = 6,
       kFindKCEvent             = 7,
       kSystemKCEvent           = 8,
       kDefaultChangedKCEvent   = 9,
       kDataAccessKCEvent       = 10
        };
   typedef UInt16 KCEvent
```

Constant descriptions

`kIdleKCEvent`
: Indicates a `NULL` event.

`kLockKCEvent`
: Indicates that the keychain was locked.

`kUnlockKCEvent`
: Indicates that the keychain was unlocked.

`kAddKCEvent`
: Indicates that an item was added to a keychain.

`kDeleteKCEvent`
: Indicates that an item was deleted from a keychain.

`kUpdateKCEvent`
: Indicates that a keychain item was updated.

`kChangeIdentityKCEvent`
: Indicates that the identity of the keychain was changed.

`kFindKCEvent`
: Indicates that a keychain item was found.

`kSystemKCEvent`
: Indicates that the event is a system event. In this case, the Keychain client can process events.

`kDefaultChangedKCEvent`
: Indicates that the default keychain has changed.

kDataAccessKCEvent
: Indicates that a process has called the function KCGetData (page 68) to access a keychain item's data.

Discussion

The KCEvent enumeration defines constants that identify the Keychain-related events your callback function wishes to receive. The Keychain Manager tests a mask that you pass in the eventMask parameter of the function KCAddCallback (page 80) to determine the data transfer events your notification callback function wishes to receive. It passes these events in the keychainEvent parameter of the function InvokeKCCallbackUPP (page 82). For a description of the Keychain-related event masks, see "Keychain Event Mask Constants" (page 97).

Keychain Event Mask Constants

Represent a mask that indicates the Keychain-related events your notification callback function will receive.

```
enum {
   kIdleKCEventMask             = 1 <<kIdleEvent,
   kLockKCEventMask             = 1 <<kLockEvent,
   kUnlockKCEventMask           = 1 <<kUnlockEvent,
   kAddKCEventMask              = 1 <<kAddEvent,
   kDeleteKCEventMask           = 1 <<kDeleteEvent,
   kUpdateKCEventMask           = 1 <<kUpdateEvent,
   kChangeIdentityKCEventMask   = 1 <<kChangeIdentityKCEvent,
   kFindKCEventMask             = 1 <<kFindKCEvent,
   kSystemEventKCEventMask      = 1 <<kSystemKCEvent
   kDefaultChangedKCEventMask   = 1 << kDefaultChangedKCEvent,
   kDataAccessKCEventMask       = 1 << kDataAccessKCEvent,
   kEveryEventKCEventMask       = 0xFFFF /* all of the above */
   };
   typedef UInt 16 KCEventMask;
```

Constant descriptions

kIdleKCEventMask
: If the bit specified by this mask is set, your callback function will be invoked during a NULL event.

kLockKCEventMask
: If the bit specified by this mask is set, your callback function will be invoked when the keychain is locked.

`kUnlockKCEventMask`
: If the bit specified by this mask is set, your callback function will be invoked when the keychain is unlocked.

`kAddKCEventMask`
: If the bit specified by this mask is set, your callback function will be invoked when an item is added to the keychain.

`kDeleteKCEventMask`
: If the bit specified by this mask is set, your callback function will be invoked when an item is removed from the keychain.

`kUpdateKCEventMask`
: If the bit specified by this mask is set, your callback function will be invoked when a keychain item is updated.

`kChangeIdentityKCEventMask`
: If the bit specified by this mask is set, your callback function will be invoked when the keychain identity is changed.

`kFindKCEventMask`
: If the bit specified by this mask is set, your callback function will be invoked when a keychain item is found.

`kSystemEventKCEventMask`
: If the bit specified by this mask is set, your callback function will be invoked when a keychain client can process events.

`kDefaultChangedKCEventMask`
: If the bit specified by this mask is set, your callback function will be invoked when the default keychain is changed.

`kDataAccessKCEventMask`
: If the bit specified by this mask is set, your callback function will be invoked when a process calls the function `KCGetData` (page 68).

`kEveryEventKCEventMask`
: If the bit specified by this mask is set, your callback function will be invoked when any of the above Keychain-related events occur.

Discussion

The `KCEventMask` enumeration defines masks your application can use to set Keychain event bits. You pass this mask in the `eventMask` parameter of the function `KCAddCallback` (page 80), thereby defining the Keychain-related events that your callback will respond to. The Keychain Manager uses this mask to test which events your callback function will handle. It passes these events in the keychainEvent parameter of the function `InvokeKCCallbackUPP` (page 82). For a description of Keychain-related events, see “Keychain Event Constants” (page 96).

Keychain Item Attribute Tag Constants

Represent tags that identify keychain item attribute values.

```
enum { /* Common attributes */
   kClassKCItemAttr             = 'clas',
   kCreationDateKCItemAttr      = 'cdat',
   kModDateKCItemAttr           = 'mdat',
   kDescriptionKCItemAttr       = 'desc',
   kCommentKCItemAttr           = 'icmt',
   kCreatorKCItemAttr           = 'crtr',
   kTypeKCItemAttr              = 'type',
   kScriptCodeKCItemAttr        = 'scrp',
   kLabelKCItemAttr             = 'labl',
   kInvisibleKCItemAttr         = 'invi',
   kNegativeKCItemAttr          = 'nega',
   kCustomIconKCItemAttr        = 'cusi',

   /* Unique Generic password attributes */
   kAccountKCItemAttr           = 'acct',
   kServiceKCItemAttr           = 'svce',
   kGenericKCItemAttr           = 'gena',

   /* Unique Internet password attributes */
   kSecurityDomainKCItemAttr    = 'sdmn',
   kServerKCItemAttr            = 'srvr',
   kAuthTypeKCItemAttr          = 'atyp',
   kPortKCItemAttr              = 'port',
   kPathKCItemAttr              = 'path',

   /* Unique Appleshare password attributes */
   kVolumeKCItemAttr            = 'vlme',
   kAddressKCItemAttr           = 'addr',
   kSignatureKCItemAttr         = 'ssig',

   /* Unique AppleShare and Internet attributes */
   kProtocolKCItemAttr          = 'ptcl',

   /* Certificate attributes */
   kSubjectKCItemAttr           = 'subj',
   kCommonNameKCItemAttr        = 'cn  ',
   kIssuerKCItemAttr            = 'issu',
   kSerialNumberKCItemAttr      = 'snbr',
kEMailKCItemAttr             = 'mail',
kPublicKeyHashKCItemAttr     = 'hpky',
kIssuerURLKCItemAttr         = 'iurl',

/* Attributes shared by keys and certificates */
kEncryptKCItemAttr           = 'encr',
kDecryptKCItemAttr           = 'decr',
kSignKCItemAttr              = 'sign',
kVerifyKCItemAttr            = 'veri',
kWrapKCItemAttr              = 'wrap',
kUnwrapKCItemAttr            = 'unwr',
kStartDateKCItemAttr         = 'sdat',
kEndDateKCItemAttr           = 'edat'
};
typedef FourCharCode KCItemAttr;
```

Constant descriptions

`kClassKCItemAttr`
: Identifies the class attribute. You use this tag to set or get a value of type `KCItemClass` that indicates whether the item is an AppleShare, Internet, or generic password, or a certificate. See "Keychain Item Class Constants" (page 104) for a description of possible values.

`kCreationDateKCItemAttr`
: Identifies the creation date attribute. You use this tag to set or get a value of type `UInt32` that indicates the date the item was created.

`kModDateKCItemAttr`
: Identifies the modification date attribute. You use this tag to set or get a value of type `UInt32` that indicates the last time the item was updated.

`kDescriptionKCItemAttr`
: Identifies the description attribute. You use this tag to set or get a value of type `string` that represents a user-visible string describing this item.

`kCommentKCItemAttr`
: Identifies the comment attribute. You use this tag to set or get a value of type `string` that represents a user-editable string containing comments for this item.

`kCreatorKCItemAttr`
: Identifies the creator attribute. You use this tag to set or get a value of type `OSType` that represents the item's creator.

`kTypeKCItemAttr`
: Identifies the type attribute. You use this tag to set or get a value of type `OSType` that represents the item’s type.

`kScriptCodeKCItemAttr`
: Identifies the script code attribute. You use this tag to set or get a value of type `ScriptCode` that represents the script code for all strings.

`kLabelKCItemAttr`
: Identifies the label attribute. You use this tag to set or get a value of type `string` that represents a user-editable string containing the label for this item.

`kInvisibleKCItemAttr`
: Identifies the invisible attribute. You use this tag to set or get a value of type `Boolean` that indicates whether the item is invisible.

`kNegativeKCItemAttr`
: Identifies the negative attribute. You use this tag to set or get a value of type `Boolean` that indicates whether there is a valid password associated with this keychain item. This is useful if your application doesn't want a password for some particular service to be stored in the keychain, but prefers that it always be entered by the user. The item (typically invisible and with zero-length data) acts as a placeholder to say “don't use me.”

`kCustomIconKCItemAttr`
: Identifies the custom icon attribute. You use this tag to set or get a value of type `Boolean` that indicates whether the item has an application-specific icon. To do this, you must also set the attribute value identified by the tag kTypeKCItemAttr to a file type for which there is a corresponding icon in the desktop database, and set the attribute value identified by the tag `kCreatorKCItemAttr` to an appropriate application creator type. If a custom icon corresponding to the item's type and creator can be found in the desktop database, it will be displayed by Keychain Access. Otherwise, default icons are used.

`kAccountKCItemAttr`
: Identifies the account attribute. You use this tag to set or get a value of type `Str63` that represents the user account. It also applies to generic and AppleShare passwords.

`kServiceKCItemAttr`
: Identifies the service attribute. You use this tag to set or get a value of type `Str63` that represents the service.

`kGenericKCItemAttr`
: Identifies the generic attribute. You use this tag to set or get a value of untyped bytes that represents a user-defined attribute.

`kSecurityDomainKCItemAttr`
: Identifies the security domain attribute. You use this tag to set or get a value of type `Str63` that represents the Internet security domain.

`kServerKCItemAttr`
: Identifies the server attribute. You use this tag to set or get a value of type `string` that represents the Internet server's domain name or IP address.

`kAuthTypeKCItemAttr`
: Identifies the authentication type attribute. You use this tag to set or get a value of type `KCAuthType` that represents the Internet authentication scheme.

`kPortKCItemAttr`
: Identifies the port attribute. You use this tag to set or get a value of type `UInt16` that represents the Internet port.

`kPathKCItemAttr`
: Identifies the path attribute. You use this tag to set or get a value of type `Str255` that represents the path.

`kVolumeKCItemAttr`
: Identifies the volume attribute. You use this tag to set or get a value of type `Str63` that represents the AppleShare volume.

`kAddressKCItemAttr`
: Identifies the address attribute. You use this tag to set or get a value of type `string` that represents the zone name, or the IP or domain name that represents the server address.

`kSignatureKCItemAttr`
: Identifies the server signature attribute. You use this tag to set or get a value of type `AFPServerSignature` (page 86) that represents the server signature block.

`kProtocolKCItemAttr`
: Identifies the protocol attribute. You use this tag to set or get a value of type `KCProtocolType` that represents the Internet protocol.

`kSubjectKCItemAttr`
: Identifies the subject attribute. You use this tag to set or get `DER`-encoded data that represents the subject distinguished name.

`kCommonNameKCItemAttr`
: Identifies the common name attribute. You use this tag to set or get a `UTF8`-encoded string that represents the common name.

`kIssuerKCItemAttr`
: Identifies the issuer attribute. You use this tag to set or get a `DER`-encoded data that represents the issuer distinguished name.

`kSerialNumberKCItemAttr`
: Identifies the serial number attribute. You use this tag to set or get a `DER`-encoded data that represents the serial number.

`kEMailKCItemAttr`
: Identifies the email attribute. You use this tag to set or get an `ASCII`-encoded string that represents the issuer 's email address.

`kPublicKeyHashKCItemAttr`
: Identifies the public key hash attribute. You use this tag to set or get a value of type `KCPublicKeyHash` (page 89) that represents the hash of the public key.

`kIssuerURLKCItemAttr`
: Identifies the issuer URL attribute. You use this tag to set or get an `ASCII`-encoded string that represents the URL of the certificate issuer.

`kEncryptKCItemAttr`
: Identifies the encrypt attribute. You use this tag to set or get a value of type `Boolean` that indicates whether the item can encrypt.

`kDecryptKCItemAttr`
: Identifies the decrypt attribute. You use this tag to set or get a value of type `Boolean` that indicates whether the item can decrypt.

`kSignKCItemAttr`
: Identifies the sign attribute. You use this tag to set or get a value of type `Boolean` that indicates whether the item can sign.

`kVerifyKCItemAttr`
: Identifies the verify attribute. You use this tag to set or get a value of type `Boolean` that indicates whether the item can verify.

`kWrapKCItemAttr`
: Identifies the wrap attribute. You use this tag to set or get a value of type `Boolean` that indicates whether the item can wrap.

`kUnwrapKCItemAttr`
: Identifies the unwrap attribute. You use this tag to set or get a value of type `Boolean` that indicates whether the item can unwrap.

`kStartDateKCItemAttr`
: Identifies the start date attribute. You use this tag to set or get a value of type `UInt32` that indicates the start date.

`kEndDateKCItemAttr`
: Identifies the end date attribute. You use this tag to set or get a value of type `UInt32` that indicates the end date.

Discussion

The `KCItemAttr` enumeration defines the Apple-defined tag constants that identify keychain item attribute values. Your application can use one of these tags in the `tag` field of the structure `KCAttribute` (page 86) to identify the keychain item attribute value you wish to set or retrieve. Your application can create application-defined tags of type `KCAttrType` (page 87).

Keychain Item Class Constants

Identify the type of the keychain item you want to create.

```
enum {
   kCertificateKCItemClass          = 'cert',
   kAppleSharePasswordItemClass     = 'ashp',
   kInternetPasswordItemClass       = 'inet',
   kGenericPasswordItemClass        = 'genp'
   };
   typedef FourCharCode KCItemClass;
```

Constant descriptions

`kCertificateKCItemClass`
: Specifies that the item is a certificate.

`kAppleSharePasswordItemClass`
: Specifies that the item is an AppleShare password.

`kInternetPasswordItemClass`
: Specifies that the item is an Internet password.

`kGenericPasswordItemClass`
: Specifies that the item is a generic password.

Discussion

The `KCItemClass` enumeration defines constants your application can use to specify the type of the keychain item you wish to create. You can pass a constant of this type in the `itemClass` parameter of the function <codeXRefText>KCNewItem (page 63). You can use these constants to specify the value of a keychain item identified by the tag constant `kClassKCItemAttr`, described in "Keychain Item Attribute Tag Constants" (page 99).

Keychain Protocol Type Constants

Represent the type of protocol to use in storing and retrieving Internet passwords.

```
enum {
   kKCProtocolTypeFTP            = 'ftp ',
   kKCProtocolTypeFTPAccount     = 'ftpa',
   kKCProtocolTypeHTTP           = 'http',
   kKCProtocolTypeIRC            = 'irc ',
   kKCProtocolTypeNNTP           = 'nntp',
   kKCProtocolTypePOP3           = 'pop3',
   kKCProtocolTypeSMTP           = 'smtp',
   kKCProtocolTypeSOCKS          = 'sox ',
   kKCProtocolTypeIMAP           = 'imap',
   kKCProtocolTypeLDAP           = 'ldap',
   kKCProtocolTypeAppleTalk      = 'atlk',
   kKCProtocolTypeAFP            = 'afp ',
   kKCProtocolTypeTelnet         = 'teln'
   };
   typedef FourCharCode KCProtocolType;
```

Constant descriptions

`kKCProtocolTypeFTP`	Specifies the File Transfer Protocol.
`kKCProtocolTypeFTPAccount`	Specifies the File Transfer Protocol Account.
`kKCProtocolTypeHTTP`	Specifies the HyperText Transfer Protocol.
`kKCProtocolTypeIRC`	Specifies the Internet Relay Channel Protocol.
`kKCProtocolTypeNNTP`	Specifies the Network News Transfer Protocol.
`kKCProtocolTypePOP3`	Specifies the Post Office 3 Protocol.

`kKCProtocolTypeSMTP`
: Specifies the Simple Mail Transfer Protocol.

`kKCProtocolTypeSOCKS`
: Specifies the Secure Proxy Server Protocol.

`kKCProtocolTypeIMAP`
: Specifies the Internet Message Access Protocol.

`kKCProtocolTypeLDAP`
: Specifies the Lightweight Directory Access Protocol.

`kKCProtocolTypeAppleTalk`
: Specifies the AppleTalk Protocol.

`kKCProtocolTypeAFP`
: Specifies the AppleTalk File Protocol.

`kKCProtocolTypeTelnet`
: Specifies the Telnet Protocol.

Discussion

The `KCProtocolType` enumeration defines constants you can use to identify the type of authentication to use in storing and retrieving Internet passwords. You can pass a constant of this type in the `protocol` parameter of the functions `KCAddInternetPassword` (page 52), `KCAddInternetPasswordWithPath` (page 53), `KCFindInternetPassword` (page 55), and `KCFindInternetPasswordWithPath` (page 57).

Keychain Status Mask Constants

Represent a mask identifying the status of a keychain.

```
enum{
       kUnlockStateKCStatus     = 1,
       kRdPermKCStatus          = 2,
       kWrPermKCStatus          = 4
   };
   typedef UInt32 KCStatus;
```

Constant descriptions

`kUnlockStateKCStatus`
: If the bit specified by this mask is set (bit 0), the keychain is unlocked.

`kRdPermKCStatus`
: If the bit specified by this mask is set (bit 1), the keychain is unlocked with read permission.

`kWrPermKCStatus`

If the bit specified by this mask is set (bit 2), the keychain is unlocked with write permission.

Discussion

The `KCStatus` enumeration defines masks your application can use to determine the status of a keychain. This mask is passed back in the `keychainStatus` parameter of the function `KCGetStatus` (page 43).

Verification Criteria Constants

Represent the verification criteria to use in selecting certificates.

```
enum {
   kPolicyKCStopOn          = 0,
   kNoneKCStopOn            = 1,
   kFirstPassKCStopOn       = 2,
   kFirstFailKCStopOn       = 3
   };
   typedef UInt16 KCVerifyStopOn;
```

Constant descriptions

`kPolicyKCStopOn`

Indicates that the function `KCChooseCertificate` (page 77) should use the trust policy options currently in effect.

`kNoneKCStopOn`

Indicates that `KCChooseCertificate` completes after examining all available certificates.

`kFirstPassKCStopOn`

Indicates that `KCChooseCertificate` when one certificate meeting the verification criteria is found.

`kFirstFailKCStopOn`

Specifies that `KCChooseCertificate` completes when one certificate that fails to meet the verification criteria is found.

Discussion

The `KCVerifyStopOn` enumeration defines constants your application can use to identify the verification criteria to use in selecting ceritifcates. You can pass a constant of this type in the `stopOn` parameter of the function `KCChooseCertificate` (page 77).

Keychain Manager Result Codes

Most Keychain Manager functions return result codes of type `OSStatus`. This includes general result codes such as `noErr`, indicating that the function completed successfully, and `paramErr`, indicating that you passed an invalid parameter. In addition, many Keychain functions may return result codes that are not Keychain-specific result codes. For example, a file system or network error may be returned if your application has no write access to a storage device.

The result codes specific to the Keychain Manager are listed in Table 4-1 (page 108). In some cases, the function result section for a particular function provides more detail about the meaning of the result code specific to that function.

Table 4-1 Keychain Manager result codes

Result code constant	Value	Description
`errKCNotAvailable`	-25291	Indicates that the Keychain Manager was not loaded.
`errKCReadOnly`	-25292	Returned by the function `KCCopyItem` (page 72) to indicate that the keychain file is read-only and cannot be edited.
`errKCAuthFailed`	-25293	Returned by the function `KCUnlock` (page 44) to indicate that the authentication failed (too many unsuccessful retries).
`errKCNoSuchKeychain`	-25294	Returned by the functions `KCUnlock` (page 44), `KCSetDefaultKeychain` (page 41), `KCGetStatus` (page 43), and `KCGetIndKeychain` (page 46) to indicate that the specified keychain was not found.

Table 4-1 Keychain Manager result codes

Result code constant	Value	Description
`errKCInvalidKeychain`	-25295	Returned by the functions `KCUnlock` (page 44), `KCSetDefaultKeychain` (page 41), `KCGetStatus` (page 43), `KCGetKeychainName` (page 43), `KCChangeSettings` (page 42), and `KCCreateKeychain` (page 39) to indicate that the keychain is not valid.
`errKCDuplicateKeychain`	-25296	Returned by the function `KCCreateKeychain` (page 39) to indicate that your application tried to create a keychain that already exists.
`errKCDuplicateCallback`	-25297	Returned by the function `KCAddCallback` (page 80) to indicate that your callback function was already registered.
`errKCInvalidCallback`	-25298	Returned by the function `KCRemoveCallback` (page 81) to indicate that the callback function was not previously registered.
`errKCDuplicateItem`	-25299	Returned by the functions `KCAddAppleSharePassword` (page 48), `KCAddInternetPassword` (page 52), `KCAddInternetPasswordWithPath` (page 53), `KCAddGenericPassword` (page 60), and `KCAddItem` (page 70) to indicate that you tried to add an existing keychain item to the keychain.
`errKCItemNotFound`	-25300	Returned by the functions `KCFindAppleSharePassword` (page 49), `KCFindInternetPassword` (page 55), `KCFindInternetPasswordWithPath` (page 57), `KCFindGenericPassword` (page 61), `KCFindNextItem` (page 75), and `KCFindFirstItem` (page 74) to indicate that no matching item was found.

Table 4-1 Keychain Manager result codes

Result code constant	Value	Description
`errKCBufferTooSmall`	-25301	Returned by the functions `KCFindAppleSharePassword` (page 49), `KCFindInternetPassword` (page 55), `KCFindInternetPasswordWithPath` (page 57), `KCFindGenericPassword` (page 61), `KCGetAttribute` (page 66), `KCGetData` (page 68), and `KCFindX509Certificates` (page 76) to indicate that the buffer was not large enough to contain the password data.
`errKCDataTooLarge`	-25302	Returned by the functions `KCAddAppleSharePassword` (page 48), `KCAddInternetPassword` (page 52), `KCAddInternetPasswordWithPath` (page 53), `KCAddGenericPassword` (page 60), `KCSetAttribute` (page 65), and `KCSetData` (page 67) to indicate that the data is too large.
`errKCNoSuchAttr`	-25303	Returned by the functions `KCSetAttribute` (page 65), `KCGetAttribute` (page 66), and `KCFindFirstItem` (page 74) to indicate that no such attribute exists.
`errKCInvalidItemRef`	-25304	Returned by the functions `KCSetAttribute` (page 65), `KCGetAttribute` (page 66), `KCSetData` (page 67), `KCGetData` (page 68), `KCAddItem` (page 70), `KCDeleteItem` (page 70), `KCUpdateItem` (page 71), `KCCopyItem` (page 72), and `KCGetKeychain` (page 72) to indicate that the keychain item reference is invalid.
`errKCInvalidSearchRef`	-25305	Returned by the functions `KCFindNextItem` (page 75) and `KCReleaseSearch` (page 76) to indicate that the specified search reference is invalid.
`errKCNoSuchClass`	-25306	Returned by the function `KCCopyItem` (page 72) to indicate that the item class does not exist.

Table 4-1 Keychain Manager result codes

Result code constant	Value	Description
`errKCNoDefaultKeychain`	-25307	Returned by the functions `KCChangeSettings` (page 42), `KCSetDefaultKeychain` (page 41), `KCGetDefaultKeychain` (page 41), `KCAddAppleSharePassword` (page 48), `KCAddInternetPassword` (page 52), `KCAddInternetPasswordWithPath` (page 53), `KCAddGenericPassword` (page 60), `KCFindAppleSharePassword` (page 49), `KCFindInternetPassword` (page 55), `KCFindInternetPasswordWithPath` (page 57), `KCFindGenericPassword` (page 61), `KCCopyItem` (page 72), `KCAddItem` (page 70), `KCDeleteItem` (page 70), `KCUpdateItem` (page 71), `KCFindNextItem` (page 75), `KCFindFirstItem` (page 74), and `KCFindX509Certificates` (page 76) to indicate that there is no default keychain.
`errKCInteractionNotAllowed`	-25308	Returned by the functions `KCCreateKeychain` (page 39), `KCChangeSettings` (page 42), `KCUnlock` (page 44), and `KCGetData` (page 68) (the latter two only when the Unlock Dialog and Allow Access dialogs are needed) to indicate that there is no start-up keychain.
`errKCReadOnlyAttr`	-25309	Returned by the function `KCSetAttribute` (page 65) to indicate that the keychain item attribute is read-only.
`errKCWrongKCVersion`	-25310	Indicates that the wrong version of Keychain Manager is installed to perform this operation.
`errKCKeySizeNotAllowed`	-25311	Indicates that the key size is illegal.
`errKCNoStorageModule`	-25312	Returned by functions that prompts the loading of the Keychain Manager to indicate that the storage module is not found.
`errKCNoCertificateModule`	-25313	Returned when a function is required for a certificate and the certificate module is not found.

Table 4-1 Keychain Manager result codes

Result code constant	Value	Description
`errKCNoPolicyModule`	-25314	Returned when a function is required for a trust policy and the policy module is not found.
`errKCInteractionRequired`	-25315	Returned by the function `KCUnlock` (page 44) to indicate that user interaction is required for this operation.
`errKCDataNotAvailable`	-25316	Indicates that the requested data is not available.
`errKCDataNotModifiable`	-25317	Returned by the functions `KCSetData` (page 67) and `KCGetData` (page 68) to indicate that the data cannot be modified.
`errKCCreateChainFailed`	-25318	Returned by the functions `KCChooseCertificate` (page 77) and `KCFindX509Certificates` (page 76) to indicate that the attempt to create a new keychain failed.

CHAPTER 5

API and Document Revision History

This section describes changes to the Keychain Manager API from version 1.0.1 to 2.0, as well as a release history of this document.

The Keychain Manager 1.0.1 SDK was the first version of Keychain released to developers. Between Keychain Manager 1.0.1 and 2.0, a number of significant changes have been made to the API in order to accommodate additional features in the Kaychain software. In general, applications which only make use of the high-level functions provided in Keychain 1.0.1 will run unmodified in 2.0. Applications that call lower-level Keychain functions in order to manipulate keychain items or their attributes, or change keychain information may need to be revised to be compatible with Keychain Manager 2.0.

If your application uses the Keychain Manager 1.0.1 SDK, you should see “Keychain Manager 2.0 API Changes” for specific information about API changes since 1.0.1. It is available as part of the Security SDK at the Apple Developer website at

<http://developer.apple.com/>

This document has had the following releases:

Table 5-1 Implementing Security Features With Keychain revision history

Publication date	Notes
May 17, 2000	First public release of document, expanded and revised for Keychain Manager 2.0. Includes concepts, tasks, and reference material. New document title: Implementing Security Features With Keychain.

Table 5-1 Implementing Security Features With Keychain revision history

Publication date	**Notes**
Aug. 5, 1998	Updated draft of Keychain 1.0 API documentation. This document was distributed in limited release as a seed draft. Document title: Enabling Secure Storage With the Keychain Manager
May 28, 1998	First draft of Keychain 1.0 API documentation. This document was distributed in limited release as a seed draft. Document title: Simplifying Password Access With the Keychain Manager