Authentication Manager

This chapter describes the AOCE Authentication Manager, which provides authentication services for users of PowerShare catalog servers. Providers of other AOCE-compatible catalog servers can also use the Authentication Manager and the AppleTalk Secure Data Stream Protocol (ASDSP) to provide authentication services for users of their catalog servers. The services provided by the Authentication Manager ensure both ends of a connection that the entity on the other end is who or what it claims to be. The Authentication Manager does not encrypt data or guarantee the integrity of transmitted data. For other security services, see the chapter "Digital Signature Manager" in this book.

The Authentication Manager application programming interface (API) provides the tools you need to implement an authenticated connection between two entities. Also, the API includes a function that provides a common server-based time service.

The Authentication Manager provides low-level functions that are called by the AOCE Collaboration package, the AOCE Collaboration toolbox, the PowerTalk Key Chain, and the PowerShare Admin program.

An application running in the background might call the Authentication Manager to get a local identity or a specific identity. You might want to add your application to the local-identity notification queue, so that the Authentication Manager calls your notification routine when the local identity is locked or unlocked or when the local-identity name is changed.

You must read this chapter if you want to create your own authentication service using AOCE functions. For example, if you want to authenticate connections between users who are not connected over an AppleTalk network, you can use the Authentication Manager functions described in this chapter.

This chapter starts with a brief introduction to authentication, including an introduction to the role of servers in authentication. The chapter then presents information to help you use the Authentication Manager functions to

- generate and use encryption keys
- create and use authentication identities
- acquire and use credentials for mutual verification of users' identities
- generate proxies and use them
- resolve creation IDs for records
- obtain the universal coordinated time
- implement your own challenge process for authenticating two entities

The language specific to this technology is defined as the concepts are introduced in the chapter.

For a general overview of AOCE services, see the chapter "Introduction to Apple Open Collaboration Environment" in this book.

Introduction to Authentication

To avoid fraud or impersonation, two users or services communicating over a network may need to identify each other conclusively. For example, a user may want to verify that a piece of electronic mail came from the sender named in the letter. In the world of networking, verification of the identity of an entity on a network or of one end of a communication link is called ***authentication***.

The authentication process involves the exchange between two parties of a sequence of messages referred to as *challenges* and *replies*. The Authentication Manager uses the Data Encryption Standard (DES, a symmetric private-key encryption algorithm that uses the same key for encryption and decryption) and a secret key derived from the user's password to encrypt each challenge or reply message. The authentication server knows the keys of both ends of the connection. Keys are discussed in the next section.

These are the basic assumptions fundamental to authentication:

- Each user or service has a key, and that key is known only to the user and the authentication server.
- The authentication server is trusted to reveal the secret key to no one.

The originator of a message is called the *initiator;* the addressee is the *recipient.* The initiator and recipient do not share a key. If they did, they could use that key to encrypt every message they exchange.

Keys

Encryption ***keys*** are numbers used by an encryption algorithm to encrypt and decrypt data. The keys of the initiator and recipient are referred to as ***client keys***. Because the authentication process requires that a trusted third party know everyone's keys, Authentication Manager functions allow you to store client keys in a server-based catalog.

The Authentication Manager uses client keys for encrypting requests to the server and for encrypting the response the server returns to an initiator. The server also uses client keys to verify that a user typed his or her password correctly.

During the authentication process, the authentication server creates a unique time-limited ***session key***, encrypts it, and transmits it to the initiator, who sends it to the recipient. The initiator and recipient use the session key to exchange challenges and replies. The section "Steps in the Authentication Process" beginning on page 9-401 describes the use of client keys and session keys.

Credentials

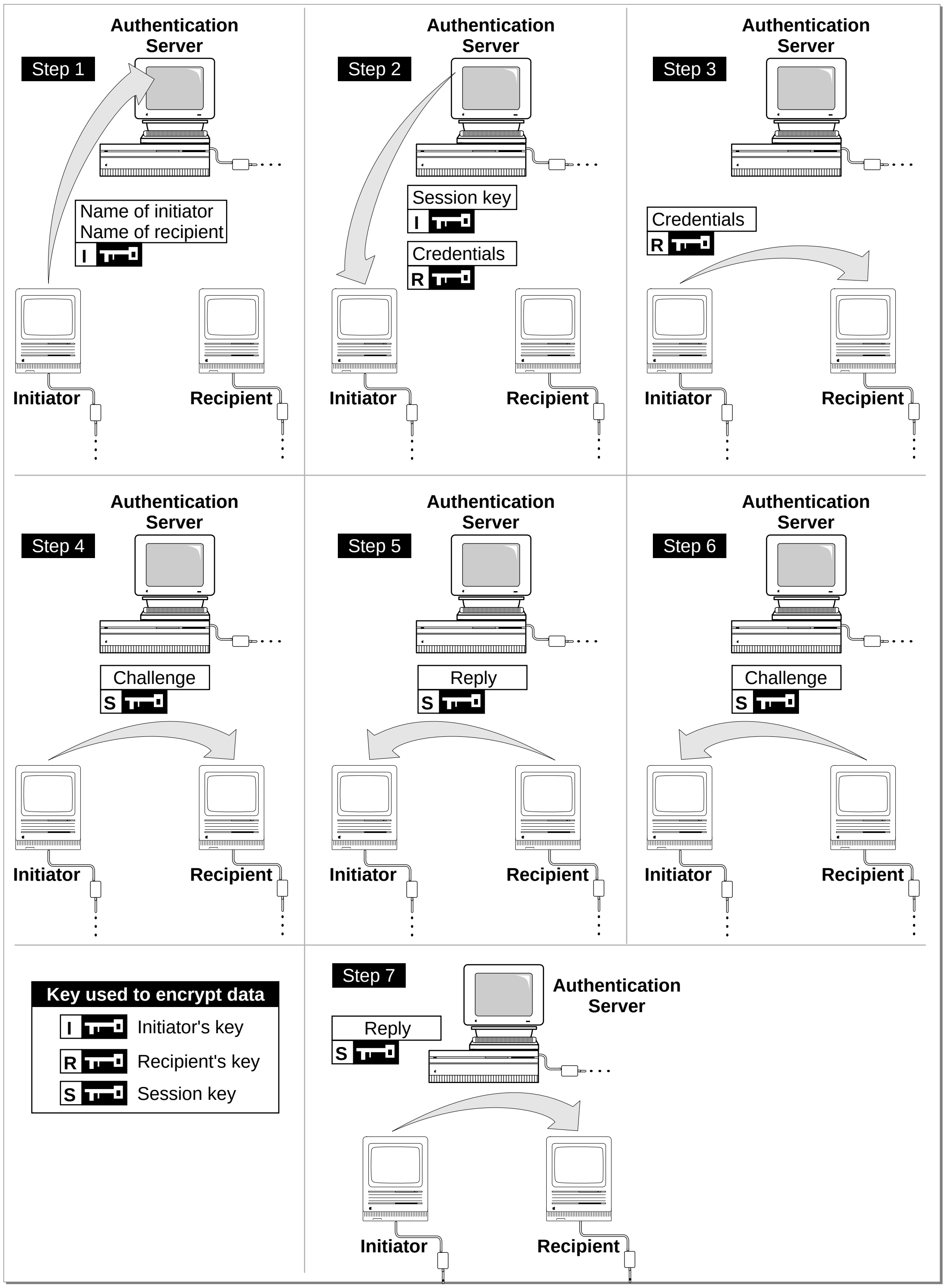
Credentials consist of an identifier for the initiator and a session key, encrypted in the key of the recipient. The initiator requests credentials from the authentication server and sends them to the recipient. With these, the recipient can determine which initiator wants to make an authenticated connection and can obtain the session key needed to complete the authentication process. Because the credentials are encrypted in the recipient's client key, only the intended recipient can use them, and the initiator cannot alter them. Therefore, the initiator can be sure that anyone responding with the correct session key is the intended recipient, and the recipient can be sure of the identity of the initiator.

Credentials are valid only for a particular initiator and recipient and only for a specific time period. After that time period, they cannot be used to establish a connection. However, once a communication stream is open and authenticated, the two ends of a connection can elect to maintain the connection even after the credentials have expired.

Steps in the Authentication Process

The authentication process consists of two phases: the *precontact phase* and the *challenge phase*. Figure 9-1 on page 9-402 shows the authentication process; in this figure, step 1 and step 2 represent the precontact phase, and the remaining steps represent the challenge phase of authentication. In Figure 9-1, For each step in the process, the figure shows what key was used to encrypt the data, who sends the data and to whom, and the nature of the data sent.

Figure 9-1 The authentication process



Here is what happens in the precontact phase of authentication:

1. The initiator encrypts both the name of the initiator and the name of the recipient in the initiator's client key and asks the server for credentials.
2. The server returns two quantities to the initiator: a session key and a credentials block. The session key is encrypted in the initiator's key. The credentials block is encrypted in the recipient's key so that not even the initiator can see what is in it.

Receipt of the credentials by the initiator completes the precontact phase of authentication. Next, the initiator can either use AppleTalk Secure Data Stream Protocol (ASDSP) to perform the challenge phase of authentication or else implement the challenge phase as described below. See the chapter "AppleTalk Data Stream Protocol" in *Inside Macintosh: Networking* for a discussion of ASDSP.

3. The initiator sends the credentials block to the recipient. This credentials block is encrypted in the recipient's key and contains the name of the initiator and a copy of the session key.

Now both the initiator and the recipient have a copy of the same session key. They now exchange challenges and replies to verify that each has the same session key.

4. The initiator selects a random number, encrypts it with the session key, and sends it to the recipient as a challenge.
5. The recipient decrypts the challenge, adds 1 to the number, encrypts the sum with the session key, and sends the new encrypted number to the initiator as a reply.

Because only the intended recipient can decrypt the credentials and therefore obtain the session key, the initiator has now established that the challenge was not intercepted by an impostor. The recipient must now issue a challenge to ensure that the initiator is truly the entity identified in the credentials.

6. The recipient selects a new random number, encrypts it with the session key, and sends it to the initiator as a challenge.
7. The initiator decrypts the number, adds 1, encrypts the sum with the session key, and sends it as a reply.

After two entities desiring a connection successfully complete this authentication process, they are ready to exchange authenticated messages. If you use ASDSP as the transport mechanism between an initiator and a recipient, the challenge phase of the authentication process is handled by the ASDSP function. If you are using another transport protocol, such as TCP/IP (Transmission Control Protocol/Internet Protocol), you can implement steps 4 through 7 of the authentication process using Authentication Manager functions described in "Non-ASDSP Authentication Utilities" beginning on page 9-450.

Identities

An ***identity***, sometimes referred to as an *authentication identity*, is a number used as shorthand for the name and key of a user or service. Many AOCE functions require an identity to determine if the initiator is authorized to make a particular service request. There are two types of authentication identities: *local identities* and *specific identities*.

Whereas a local identity is associated with a particular computer, a specific identity is associated with a particular server or service. In most cases you use the local identity when you call an AOCE function, except when providing access to a service on behalf of someone other than the principal user of the computer. Local identities and specific identities are discussed in the following sections.

Local Identities

Because a user may have multiple "accounts" for a variety of applications or services, the PowerTalk system software provides a Setup catalog that contains (in encrypted form) the names and passwords for the services available to the user. A ***local identity*** is a number used as shorthand for the name and password associated with the user of a particular computer. This local identity is a "master" identity because it provides access to all catalogs and services in the PowerTalk Setup catalog without requiring each service's password individually. Any AOCE function that requires an identity as input can use the local identity.

The Standard Catalog Package function, `SDPPromptForID`, described in the chapter "Standard Catalog Package" in this book, prompts the user for his or her name and password and uses this information to generate the local identity.

A background application can obtain the local identity generated by the `SDPPromptForID` function by calling the Authentication Manager's `AuthGetLocalIdentity` function, described on page 9-424. If a local identity is not set up, you can install your application in a notification queue, so that the application is notified when the local identity is created or unlocked.

By supplying a valid local identity to any AOCE function that requires an identity parameter, you tell the AOCE toolbox what user is requesting the service. The toolbox prepares an authenticated stream to the server, and during this process the server learns the name of the user. Then the server checks the access controls for the user represented by the identity to ensure that the user has the privileges necessary to access the requested function. If the access controls are sufficient, the AOCE software provides the requested service. Otherwise, you receive a result code stating that the user's access rights are insufficient. Access controls are discussed in the chapter "Catalog Manager" in this book.

The functions you can use to manage local identities are described in "Local Identity Management" beginning on page 9-424.

Locking and Unlocking Local Identities

The PowerTalk system software gives users the option of protecting their accounts from unauthorized access. To do so, the user chooses Lock Key Chain from the Special menu of the Finder or sets the PowerTalk Setup control panel to lock the Key Chain after some specified period of inactivity. Upon returning, the user chooses Unlock Key Chain from the Finder's Special menu and is prompted for a password. You can also lock and unlock the local identity from within your application.

If the local identity is locked, it is the responsibility of your application to disable its own services appropriately. For example, if you are designing a mail application, you may want it to continue receiving mail even when the local identity is locked but would probably not want to allow users to read mail that has been received.

Local Identity Status Notification

If your application needs to enable or disable features based on whether the local identity is unlocked, you may want to be notified of changes in the status of the local identity. If so, you can add your application to a notification queue. The applications in this queue are notified when the local identity is unlocked or locked. Through the notification queue, you can deny locking of the local identity when your application is in use. For instance, you might want to deny locking when your application is engaged in some process that would be seriously disrupted if the lock function succeeded.

Specific Identities

To provide a service to a user other than the principal user of a computer, you can use a specific identity rather than the local identity. The ***specific identity*** is a number used as shorthand for the name and key of the alternate user. You can use the specific identity in any AOCE function that requires an identity.

The Standard Catalog Package function `SDPPromptForID` prompts a user for a name and password and returns a specific identity. The `SDPPromptForID` function is described in the chapter “Standard Catalog Package” in this book.

Guest Access

When your application needs to accommodate users with no accounts on the computer or server, you can specify a “guest identity” by using the value 0 for the identity parameter in AOCE functions.

The PowerTalk Setup Catalog

The AOCE Catalog Manager defines a special personal catalog called the ***PowerTalk Setup catalog***, which contains information about the catalogs and other services that are available to the principal user of the computer. The PowerTalk Setup catalog is stored on the user’s local disk. The records in the PowerTalk Setup catalog represent such entities as PowerShare catalogs, external catalogs, and catalog service access modules (CSAMs). Catalogs and CSAMs represented by records in the PowerTalk Setup catalog are said to be “listed in the PowerTalk Setup catalog.” The contents of the Setup catalog and the process of adding a CSAM or mail service access module (MSAM) to the Setup catalog are described in the chapter “Service Access Module Setup” in *Inside Macintosh: AOCE Service Access Modules*.

You can use the functions described in “PowerTalk Setup Catalog Management” beginning on page 9-457 to set up, change, remove, or get information about items in the PowerTalk Setup catalog.

Proxies

A ***proxy*** allows an alternate entity to be authenticated as the user for a limited time. It is a privilege provided to an ***intermediary***: a representative of the user or service. The intermediary uses the proxy to obtain the credentials needed to complete the authentication process. The proxy gives the intermediary access to a particular recipient to perform some task on behalf of an initiator.

For example, suppose a user of your application plans to be away from the computer but wants to back up some data when the server is not busy. In this case, your application can request a proxy for the user. You may assign the proxy to an intermediary, who can do the backup. With this proxy, the intermediary obtains credentials from the server and then uses them to create an authenticated connection in the usual way. Functions you can use to create and use proxies are described in "Credentials Management" beginning on page 9-439.

About the Authentication Manager

The Authentication Manager, the Digital Signature Manager, the Catalog Manager, and the Interprogram Messaging Manager together constitute the fundamental services of the AOCE system software. The Standard Catalog Package and the Standard Mail Package provide high-level interfaces to the Authentication Manager.

The Authentication Manager is a collection of functions that runs on the user's computer and communicates with the authentication server to set up authenticated connections.

The Authentication Manager includes routines that provide the following services:

- key management: translating passwords to keys and adding, changing, and deleting keys in the server
- local identity management: determining the local identity for a computer; locking, unlocking, creating, changing, and removing local identities; and adding applications to and removing them from a notification queue for changes in the status of the local identity
- specific identity management: binding, unbinding, and getting information about specific identities
- credentials management: obtaining and using credentials and making and using proxies
- resolution of creation IDs: resolving creation IDs when multiple records have the same name and type
- time service: obtaining the universal coordinated time
- non-ASDSP authentication utilities: performing authentication as a step-by-step process
- PowerTalk Setup catalog management: setting up, changing, removing, and getting information about catalogs in the PowerTalk Setup catalog

Using the Authentication Manager

This section discusses the techniques you can use to perform tasks related to authentication. You can use the techniques in this section to

- perform the authentication process for initiators and recipients using ASDSP
- perform the precontact phase and challenge process of authentication for initiators and recipients using a different transport mechanism
- use a proxy in either of the above authentication processes
- monitor the status of access to the PowerTalk Setup catalog by installing your application in a notification queue

For more detailed descriptions of the routines described in this section, see "Authentication Manager Functions" beginning on page 9-416.

Determining Whether the Collaboration Toolbox Is Available

Before calling any of the Authentication Manager functions, you should verify that the Collaboration toolbox is available by calling the `Gestalt` function with the selector `gestaltOCEToolboxAttr`. If the Collaboration toolbox is present but not running (for example, if the user deactivated it from the PowerTalk Setup control panel), the `Gestalt` function sets the bit `gestaltOCETBPresent` in the `response` parameter. If the Collaboration toolbox is running and available, the function sets the bit `gestaltOCETBAvailable` in the `response` parameter. The Gestalt Manager is described in the chapter "Gestalt Manager" of *Inside Macintosh: Operating System Utilities*.

If you want to be informed when the Authentication Manager starts up or shuts down, you can install an entry in the AppleTalk Transition Queue (ATQ). Then the AppleTalk LAP Manager calls your ATQ routine with the transition selector `ATTransAuthStart` when the Authentication Manager has finished starting up and with the selector `ATTransAuthShutdown` when the Authentication Manager has started to shut down. The ATQ is described in the chapter "Link-Access Protocol (LAP) Manager" in *Inside Macintosh: Networking*.

Determining the Version of the Authentication Manager

To determine the version of the Authentication Manager that is available, call the `Gestalt` function with the selector `gestaltOCEToolboxVersion`. The function returns the version number of the Collaboration toolbox in the low-order word of the `response` parameter. For example, a value of 0x0101 indicates version 1.0.1. If you are using the Collaboration toolbox on a computer that has a PowerShare server, the function returns the version number of the server in the high-order word of the `response` parameter. If the Collaboration toolbox or server is not present and available, the `Gestalt` function returns 0 for the relevant version number. You can use the constant `gestaltOCETB` for AOCE Collaboration toolbox version 1.0.

Authentication Using ASDSP

To establish mutual authentication between an initiator and a recipient, you use credentials that you get from the server. When you use ASDSP as the transport mechanism to complete the secure connection, you place these credentials in the appropriate field of the parameter block for the `sdspOpen` function. ASDSP is discussed in the chapter “AppleTalk Data Stream Protocol” in *Inside Macintosh: Networking*.

To get credentials, follow these steps:

1. Specify an expiration time for the `AuthGetCredentials` function (page 9-439). It is your responsibility to determine how long you want the connection to be available. Credentials are valid for at most 8 hours after they are returned to an initiator by the server. When you call the `AuthGetCredentials` function you may use the `expiry` field to specify a shorter time for credentials to be valid. Two ways to determine your expiration time are as follows:
   - Call the `AuthGetUTCTime` function (page 9-449) to get the current universal coordinated time (UTC) and an offset. Then, your expiration time is the UTC plus the amount of time, in seconds, that you want the credentials to be valid.
   - If you get credentials often, you may choose to remember the time provided by the `AuthGetUTCTime` function when you first call it and then add the results of the `GetDateTime` function to that time along with the amount of time, in seconds, that you want the credentials to be valid. Remembering the UTC makes it unnecessary to call the `AuthGetUTCTime` function each time you need credentials. The `GetDateTime` function is described in *Inside Macintosh: Operating System Utilities*.
2. Determine the initiator’s identity and the recipient’s record ID. You can use either the local identity or a specific identity for the initiator. A background application can get the local identity by calling the `AuthGetLocalIdentity` function (page 9-424). A foreground application can call the `PromptForIdentity` function, which is described in the chapter “Standard Catalog Package” in this book.

   To get a specific identity for an initiator, first call the `AuthPasswordToKey` function (page 9-417), providing the record ID and password for the initiator, to get the initiator’s client key. Then call the `AuthBindSpecificIdentity` function (page 9-435) to get the specific identity.

   You must provide your own means for obtaining the recipient’s record ID.

3. Call the `AuthGetCredentials` function to get credentials. The Authentication Manager expects you to provide the expected length of the credentials, as well as a pointer to a memory block for the credentials. A buffer three times the size of a packed record ID is usually sufficient for credentials. Use the `kPackedRecordIDMaxBytes` constant defined in the chapter "AOCE Utilities" in this book to determine the size of a packed record ID.
4. To use the ASDSP transport mechanism, call the Device Manager's `PBControl` function using the `SDSPParamBlock` parameter block defined in *Inside Macintosh: Networking*.

Authentication for Non-ASDSP Users

To establish mutual authentication between users without using ASDSP, first complete steps 1 through 3 of "Authentication Using ASDSP" on page 9-408. Then continue as indicated in the following sections.

The Initiator's Authentication Process

To complete the authentication process as an initiator, follow these steps. Note that you must devise your own protocol for exchanging the challenges and replies.

1. Call the `AuthMakeChallenge` function (page 9-451) to make a challenge. You provide a buffer and a buffer size. The buffer must be at least 8 bytes in length. The `AuthMakeChallenge` function returns the encrypted challenge in the buffer you supplied, and also returns the actual length of the challenge.
2. Send the credentials and challenge to the specified recipient, using the available transport mechanism.
3. Obtain the challenge reply from the recipient. The challenge reply includes both the reply to your challenge and a counterchallenge from the recipient (steps 5 and 6 in "Steps in the Authentication Process" beginning on page 9-401).
4. Call the `AuthVerifyReply` function (page 9-454) to verify the reply sent by the recipient and to generate a reply to the recipient's counterchallenge. You provide the session key that was supplied by the server as well as the challenge and challenge length returned by the `AuthMakeChallenge` function. You also provide the reply and reply buffer length sent by the recipient. If the `AuthVerifyReply` function finds that the recipient's reply was not valid, it returns an error and does not generate a reply to the counterchallenge.
5. If there was no error, then send the counterchallenge reply generated by the `AuthVerifyReply` function to the recipient.

The Recipient's Authentication Process

To complete authentication as a recipient, follow these steps:

1. Call the `AuthDecryptCredentials` function (page 9-455), passing it the credentials sent by the initiator. The function returns the session key, the issue and expiration times, and the record ID for the initiator. It is your responsibility to ensure that the times are acceptable for your application. Additionally, if there is an intermediary and you provide a pointer to a record ID for it, the `AuthDecryptCredentials` function provides the intermediary's record ID to you.
2. Call the `AuthMakeReply` function (page 9-452) to generate a reply to the challenge received from the initiator and to issue a challenge in return. The `challenge` pointer and `challengeLength` fields are received from the initiator and supplied to this function. The `reply` field contains the reply generated by the function and also the counterchallenge generated by the function.
3. Send this challenge reply and the counterchallenge to the initiator.
4. Obtain the counterchallenge reply from the initiator.
5. Call the `AuthVerifyReply` function to verify the reply sent by the initiator. You provide the session key that was supplied by the server with the credentials, the challenge and challenge length that you sent to the initiator, a pointer to the reply buffer, and the length of the reply.

Authentication Using a Proxy

To use a proxy to authenticate a connection, you request and receive a proxy and then give the proxy to an intermediary, who then uses the proxy to obtain credentials. After the intermediary obtains the credentials, it uses them to create an authenticated connection in the standard way, as described previously.

To obtain and use a proxy, follow these steps:

1. Call the `AuthMakeProxy` function (page 9-441). You must specify the identity of the initiator who wants to create a proxy, the record ID of the recipient with whom the intermediary wishes to communicate, and the record ID of the intermediary. Additionally, you provide times that you want the proxy to be become valid and to expire, a pointer to the buffer into which the `AuthMakeProxy` function will place the proxy, and the length of the buffer. A buffer twice the size of a packed record ID is usually sufficient for the proxy. The `kPackedRecordIDMaxBytes` constant, described in the chapter "AOCE Utilities" in this book, defines the maximum size of a packed record ID.
2. Send the proxy and the recipient record ID to the intermediary.
3. The intermediary calls the `AuthTradeProxyForCredentials` function (page 9-443), supplying the pointer to the proxy buffer and the buffer length. It also supplies its own identity and the recipient's record ID. The intermediary provides a pointer to the credentials and the expected length of the credentials. A buffer three times the size of a packed record ID is usually sufficient for credentials.

Using the Notification Queue

You can add your application's notification callback routine to a notification queue so that it is notified when the local identity is locked or unlocked. When you no longer need to know the status of the local identity, you can remove your callback routine from the notification queue. The `DoNoteQueue` routine in Listing 9-1 checks for a local identity and, if there is one, saves it in a global variable. It installs the SurfWriter application's notification callback routine in the notification queue, which informs it if the status of the local identity changes. Finally, the `DoNoteQueue` routine removes the callback routine from the queue.

If the local identity is locked and your application runs in the foreground, you should disable any functions or commands that require the user to be authenticated. You can then prompt the user to unlock or set up the local identity. If the application runs in the background, you would probably postpone some operations until the local identity is unlocked.

To install an application in or remove an application from the notification queue, you first set up the header block, as shown in the `DoInitializeASPB` function in Listing 9-1. Both the `DoInstallNotificationProc` function and `DoRemoveNotificationProc` function call the `DoInitializeASPB` function and then initialize the remaining fields for their respective functions.

The `MyNotificationProc` function in Listing 9-1 is a sample notification routine for the `AuthAddToLocalIdentityQueue` and `AuthRemoveFromLocalIdentityQueue` functions (page 9-426 and page 9-427). The `MyNotificationProc` callback routine is described on page 9-465.

In Listing 9-1, the notification routine updates a flag in the application's global data (the `identityIsLocked` field in the `MyClientData` structure) to notify the SurfWriter application when access to the PowerTalk Setup catalog is locked or unlocked. If the `identityIsLocked` field has the value `true`, the identity might be locked or might not be set up.

Listing 9-1 Using the notification queue

```
/* function to initialize header block */
pascal void DoInitializeASPB(AuthParamBlock *aspb)
   {
   *(long *)&aspb->header.serverHint = 0; /* set up serverHint */
   aspb->header.identity = 0;            /* identity setup */
   aspb->header.dsRefNum = kRefNumUnknown; /* refNum specifier */
   }
/* function to install an application's notification proc in the queue */
pascal OSErr DoInstallNotificationProc(NotificationProc notificationProc,
                                       AuthNotifications notifyFlags,
                                       StringPtr appName,
                                       long clientData)
   {
   OSErr err;
   AuthParamBlock aspb;
   DoInitializeASPB(&aspb); /* initialize header block */
   aspb.header.clientData = clientData;
   aspb.localIdentityQInstallPB.fNotificationProc=notificationProc;
   aspb.localIdentityQInstallPB.notifyFlags = notifyFlags;
   aspb.localIdentityQInstallPB.appName = appName;
   err = AuthAddToLocalIdentityQueue(&aspb, false);
   return err;
   }
/* function to remove an application’s notification proc from the queue */
pascal OSErr DoRemoveNotificationProc(NotificationProc notificationProc)
   {
   OSErr err;
   AuthParamBlock aspb;
   InitializeASPB(&aspb); /* Initialize header block */
   aspb.localIdentityQInstallPB.fNotificationProc=nNotificationProc;
   err = AuthRemoveFromLocalIdentityQueue(&aspb, false);
   return err;
   }
struct MyClientData {
   LocalIdentity localID;
   Boolean identityIsLocked;
   };

pascal OSErr MyGetLocalIdentity(LocalIdentity *localID)
   {
   OSErr err;
   AuthParamBlock aspb;
   DoInitializeASPB(&aspb); /* Initialize header block */
   err = AuthGetLocalIdentity(&aspb, false);
   if (err == noErr)
      *localID = aspb.getLocalIdentityPB.theLocalIdentity;
   return err;
   }


/* notification procedure for your application */
pascal Boolean MyNotificationProc(long clientData,
                          AuthLocalIdentityOp callValue,
                          AuthLocalIdentityLockAction actionValue,
                          LocalIdentity identity)
   {
   struct MyClientData *myClientData = (struct MyClientData *)clientData;
   if ((callValue == kAuthLockLocalIdentityOp) &&
            (actionValue == kAuthLockWillBeDone)) {
      myClientData->identityIsLocked = true;
      myClientData->localID = 0;
      }
   else
      if (callValue == kAuthUnlockLocalIdentityOp) {
         myClientData->identityIsLocked = false;
         myClientData->localID = identity;
         }
   return false;  /* the sample app never denies a lock pending */
   }


DoNoteQueue () /* using the notification queue for your application */
   {
   OSErr err;
   struct MyClientData myClientData;

   err = MyGetLocalIdentity(&myClientData.localID);
   if (err == noErr)
      myClientData.identityIsLocked = false; /* the function returned a
                                                 local identity, therefore
                                                 it's not locked */
   else {
      myClientData.identityIsLocked = true;  /* it's either not set up or
                                                 else locked */
      /* Set up the local ID if app is not in background, or else wait for
           local ID to be set up and unlocked.  If the latter, when the local
           ID is unlocked, you can get the local identity by looking at
           the localID field in MyClientData. */
      }

   err = DoInstallNotificationProc(
                  MyNotificationProc, kNotifyLockMask|kNotifyUnlockMask,
                  "\pSurfWriter", (long)&myClientData);

   /* ... perform your application's functions */
   /* If identityIsLocked is true, postpone some operations until local ID
       becomes unlocked. */

   RemoveNotificationProc(MyNotificationProc);
   }
```

Authentication Manager Reference

This section describes the data structures and routines provided by the Authentication Manager.

Data Structures

This section describes the data structures that are specific to the Authentication Manager. See the chapter “AOCE Utilities” for descriptions of other data structures that you use to provide information to or obtain information from Authentication Manager routines.

Parameter Block Header

Each Authentication Manager routine takes, as input, a pointer to a parameter block of type `AuthParamBlockPtr`. This parameter block defines a union of substructures, each of which is a parameter block for one of the Authentication Manager functions. See the descriptions of individual routines, beginning on page 9-416, for a listing of fields in the corresponding parameter blocks. Each of these parameter blocks has the following header:

```
#define AuthDirParamHeader
   Ptr            qLink;         /* reserved */
   long           reserved1;     /* reserved */
   long           reserved2;     /* reserved */
   ProcPtr        ioCompletion;  /* your completion routine */
   OSErr          ioResult;      /* result code */
   unsigned long  saveA5;        /* reserved */
   short          reqCode;       /* reserved */
   long           reserved[2];   /* reserved */
   AddrBlock      serverHint;    /* PowerShare server AppleTalk
                                    addr */
   short          dsRefNum;      /* Set to kRefNumUnknown */
   unsigned long  callID;        /* reserved */
   AuthIdentity   identity;      /* initiator’s identity */
   long           gReserved1;    /* reserved */
   long          gReserved2;    /* reserved */
   long          gReserved3;    /* reserved */
   long          clientData;    /* you define this field */
```

Field descriptions

`qLink`	Reserved.
`reserved1`	Reserved.
`reserved2`	Reserved.
`ioCompletion`	A pointer to a completion routine that you can provide. If you call an Authentication Manager routine asynchronously, it calls your completion function upon returning. Set this field to `nil` if you do not wish to provide a completion routine. The function ignores this field if you call it synchronously.
`ioResult`	The result of the routine. When you execute the routine asynchronously, the Authentication Manager sets this field to 1 as soon as it queues the routine for execution. When the routine completes execution, the Authentication Manager sets this field to the result code.
`saveA5`	Reserved.
`reqCode`	Reserved.
`reserved[2]`	Reserved.
`serverHint`	The AppleTalk address of the PowerShare server to which you want to direct your request. Normally, you specify the value 0 for all fields of this structure, and the Authentication Manager directs the request to an appropriate PowerShare server. The `AddrBlock` data structure is described in *Inside Macintosh: Networking*.
`dsRefNum`	The personal catalog reference number. Because the Authentication Manager works only with server-based catalogs, you must set this parameter to the value `kRefNumUnknown` for all Authentication Manager functions.
`callID`	Reserved.
`identity`	The authentication identity of the entity calling a function. The authentication identity can be either a local identity, a specific identity, or 0 for guest access. The PowerShare server or CSAM uses the identity to determine if the requestor has the access privileges necessary to perform the requested operation. Functions that fail because of insufficient access privileges return either the `kOCEReadAccessDenied` or `kOCEWriteAccessDenied` result code. The `AuthGetLocalIdentity` function described on page 9-424 returns the local identity, and the `AuthBindSpecificIdentity` function described on page 9-435 returns a specific identity. See the chapter "Catalog Manager" in this book for more information about access controls.
`gReserved1`	Reserved.
`gReserved2`	Reserved.
`gReserved3`	Reserved.

`clientData`	Available for your use. The Authentication Manager passes the value in this field to your completion or callback routine. If you use the same completion routine to process more than one asynchronous request, for example, your routine can use the `clientData` field to determine for which request it is processing results. You may also use this field to store a pointer to your application's private data.

The Key Structures

Keys are translated passwords used in cryptographic algorithms. See "Keys" on page 9-400. The client keys and session keys used by some Authentication Manager functions are defined by a structure of type `AuthKey`.

```
typedef unsigned long AuthKeyType;
typedef Byte RC4Key[kRC4KeySizeInBytes];
struct AuthKey {/* key type followed by its data */
   AuthKeyType keyType;
   union {
      DESKey des;
      RC4Key rc4;
   }u;
};
typedef AuthKey *AuthKeyPtr;
struct DESKey {/* A DES key is 8 bytes of data */
   unsigned long a;
   unsigned long b;
};
```

Authentication Manager Functions

This section describes functions provided by the Authentication Manager for your use. These functions make it possible for you to manage keys, local identities, specific identities, and credentials; resolve creation IDs; obtain universal coordinated time; implement non-ASDSP authentication, and manage the PowerTalk Setup catalog.

Note

As is generally true, to ensure that asynchronously called functions operate correctly, you must allocate nonrelocatable memory for all parameter blocks and any buffers required for the function. ◆

Assembly-Language Interface

To call an Authentication Manager function from assembly language, push the address of the `AuthParamBlock` parameter block and the `async` flag onto the stack using the Pascal calling convention, and place the appropriate routine selector value in register D0. Then invoke the `_oceTBDispatch` trap. Each function description includes the selector value for that function. The function returns its result code in the `ioResult` field of the parameter block.

Key Management

The Authentication Manager provides functions to

- translate a password into a key (`AuthPasswordToKey`)
- add a key to a server-based catalog (`AuthAddKey`)
- change a key in a server-based catalog (`AuthChangeKey`)
- delete a key from a server-based catalog (`AuthDeleteKey`)

The three functions that communicate with the server are subject to the access controls specified in the record of the entity for whom you're making the request. Access controls are discussed in the chapter "Catalog Manager" in this book.

Note
These functions operate only on client keys, not on session keys. Session keys are created by servers and are valid only for a limited time period. See "Keys" on page 9-400. ◆

AuthPasswordToKey

The `AuthPasswordToKey` function translates a password string into a client key.

```
pascal OSErr AuthPasswordToKey (AuthParamBlockPtr paramBlock,
                                Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`userRecord`	`RecordIDPtr`	Target's record ID
↔	`key`	`AuthKeyPtr`	Target's key
→	`password`	`RStringPtr`	Target's password

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`userRecord`	A pointer to the record ID of the user or service for which you want a client key.
`key`	A pointer to an `AuthKey` structure you allocate. The function places the key in this structure.
`password`	A pointer to the password string of the user or service whose record ID you specified in the `userRecord` parameter. Passwords must be at least 5 bytes and not more than 255 bytes.

DESCRIPTION

The `AuthPasswordToKey` function creates a new key from a new or changed password. The Authentication Manager returns the key to your local computer only; it does not store the key on the server.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$020A

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Password too long
`kOCEUndesirableKey`	–1556	Password too short or resulting key is undesirable

SEE ALSO

The `AuthKey` structure is described in "The Key Structures" on page 9-416.

The `AuthPasswordToKey` function is used in an example in "Authentication Using ASDSP" on page 9-408.

The `AuthAddKey` function is discussed next.

The `AuthChangeKey` function is described on page 9-420.

The `AuthDeleteKey` function is described on page 9-422.

AuthAddKey

The `AuthAddKey` function adds a key for an authentication client to the server-based catalog.

```
pascal OSErr AuthAddKey (AuthParamBlockPtr paramBlock,
                        Boolean async);
```

`paramBlock`
: A pointer to a parameter block.

`async`
: A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`identity`	`AuthIdentity`	Initiator’s identity
→	`userRecord`	`RecordIDPtr`	Target’s record ID
→	`userKey`	`AuthKeyPtr`	Target’s key
→	`password`	`RStringPtr`	Target’s password

See page 9-415 for descriptions of the `ioCompletion`, `ioResult`, and `identity` fields.

Field descriptions

`userRecord`
: A pointer to the record ID of the user or service whose key you are adding to a catalog.

`userKey`
: A pointer to the new key you are providing.

`password`
: A pointer to the password string of the user or service whose key you are providing. Specify `nil` for this field if you are not providing a password. If you provide a password, the Authentication Manager checks that the key was properly translated from the password before adding the key to the catalog.

DESCRIPTION

During the authentication process, the authentication server encrypts data using the keys of both the initiator and the recipient. For this reason, the server must store the key of every user of the system.

You must provide an identity to this function so that the server can check whether the caller has permission to add a key to the user’s record.

Call the `AuthPasswordToKey` function before calling the `AuthAddKey` function to obtain a key corresponding to the user’s password.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0207

RESULT CODES

`noErr`	0	No error
`kOCEAWriteAccessDenied`	–1541	Write access denied
`kOCEKeyAlreadyRegistered`	–1554	A key already exists
`kOCEMalFormedKey`	–1555	Key not derived properly from password
`kOCEUnknownID`	–1567	Identity passed is not valid
`kOCENotLocal`	–1610	Internal AOCE error
`kOCETargetDirectoryInaccessible`	–1613	Catalog server not responding
`kOCENoSuchDNode`	–1615	The dNode was not found
`kOCEBadRecordID`	–1617	Name and type incorrect for creation ID
`kOCENoSuchRecord`	–1618	No such record
`kOCEStreamCreationErr`	–1625	An error occurred in creating the stream

SEE ALSO

The use of keys in the authentication process is described in “Steps in the Authentication Process” beginning on page 9-401.

Access controls are discussed in the chapter “Catalog Manager” in this book.

Use the `AuthPasswordToKey` function (page 9-417) to create a key.

Use the `AuthChangeKey` function, described next, to replace a key already stored in the server-based catalog.

AuthChangeKey

The `AuthChangeKey` function changes a user’s key stored in a server-based catalog.

```
pascal OSErr AuthChangeKey (AuthParamBlockPtr paramBlock,
                           Boolean async);
```

`paramBlock`
: A pointer to a parameter block.

`async`
: A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`identity`	`AuthIdentity`	Initiator's identity
→	`userRecord`	`RecordIDPtr`	Target's record ID
→	`userKey`	`AuthKeyPtr`	Target's key
→	`password`	`RStringPtr`	Target's password

See page 9-415 for descriptions of the `ioCompletion`, `ioResult`, and `identity` fields.

Field descriptions

`userRecord`	A pointer to the record ID of the user or service whose changed key you are storing in a catalog.
`userKey`	A pointer to the new key you are providing.
`password`	A pointer to the password string of the user or service whose key you are providing. Specify `nil` for this field if you are not providing a password. If you provide a password, the Authentication Manager checks that the key was properly translated from the password before adding the key to the catalog.

DESCRIPTION

Call the `AuthChangeKey` function when a password has been changed and you need to store a new key in a server-based catalog. Call the `AuthPasswordToKey` function before calling the `AuthChangeKey` function to obtain a key corresponding to the new password.

You must provide an identity to this function so that the server can verify that the caller has permission to change a key in the user's record.

SPECIAL CONSIDERATIONS

If you change a key for a user or service and later attempt to use a local or specific identity that was created using the old key, the function may fail. It is important to update identities when changes are made to the passwords and therefore to the keys. Before executing some functions, the Collaboration toolbox communicates with the server to check identities and keys relative to each other.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0208

RESULT CODES

`noErr`	0	No error
`kOCEAWriteAccessDenied`	–1541	Write access denied
`kOCENoKeyFound`	–1550	No key was found
`kOCEMalFormedKey`	–1555	Key not derived properly from password
`kOCEUnknownID`	–1567	Identity passed is not valid
`kOCENotLocal`	–1610	Internal AOCE error
`kOCETargetDirectoryInaccessible`	–1613	Catalog server not responding
`kOCENoSuchDNode`	–1615	The dNode was not found
`kOCEBadRecordID`	–1617	Name and Type incorrect for creation ID
`kOCENoSuchRecord`	–1618	No such record
`kOCEStreamCreationErr`	–1625	An error occurred in creating the stream

SEE ALSO

Use the `AuthBindSpecificIdentity` function (page 9-435) to update an identity when you change a key.

The `AuthPasswordToKey` function is described on page 9-417.

The `AuthAddKey` function is discussed on page 9-419.

The `AuthDeleteKey` function is described next.

AuthDeleteKey

Call the `AuthDeleteKey` function to delete a key for a specified authentication client from the server-based catalog.

```
pascal OSErr AuthDeleteKey (AuthParamBlockPtr paramBlock,
                            Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`identity`	`AuthIdentity`	Initiator’s identity
→	`userRecord`	`RecordIDPtr`	Target’s record ID

See page 9-415 for descriptions of the `ioCompletion`, `ioResult`, and `identity` fields.

Field descriptions

`userRecord`	A pointer to the record ID of the user or service whose key is to be deleted.

DESCRIPTION

Call the `AuthDeleteKey` function to remove a `key` from the server-based catalog.

If you wish a new key to take the place of the deleted one in the server-based catalog, you can call the `AuthPasswordToKey` function and then the `AuthAddKey` function.

SPECIAL CONSIDERATIONS

When you remove a key for a user or service from the server-based catalog, the Authentication Manager can no longer create an authentication identity for that user or service, build credentials, or have others build credentials to authenticate connections to the user or service.

If you remove a key for a user and then later attempt to use a local or specific identity that was created using the key, the function may fail. It is important to update identities when changes are made to passwords and therefore to keys. Identities and keys are checked relative to each other before some functions are allowed.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0209

RESULT CODES

`noErr`	0	No error
`kOCEAWriteAccessDenied`	–1541	Write access denied
`kOCENoKeyFound`	–1550	No key was found
`kOCEUnknownID`	–1567	Identity passed is not valid
`kOCENotLocal`	–1610	Internal AOCE error
`kOCETargetDirectoryInaccessible`	–1613	Catalog server not responding
`kOCENoSuchDNode`	–1615	The dNode was not found
`kOCEBadRecordID`	–1617	Name and type incorrect for creation ID
`kOCENoSuchRecord`	–1618	No such record
`kOCEStreamCreationErr`	–1625	An error occurred in creating the stream

SEE ALSO

The `AuthPasswordToKey` function is described on page 9-417.

The `AuthAddKey` function is discussed on page 9-419

The `AuthChangeKey` function is described on page 9-420.

Use the `AuthBindSpecificIdentity` function (page 9-435) to update an identity when you change a key.

Local Identity Management

A local identity provides transparent access to the PowerTalk Setup catalog: it gives the user access to all catalogs and services in the PowerTalk Setup catalog without the user having to log on to each one individually. Any AOCE Catalog Manager or Authentication Manager function that requires an `identity` parameter can use a local identity. See "Local Identities" on page 9-404 for a discussion of local identities.

The Authentication Manager provides functions that you can use to

- get the local identity number (`AuthGetLocalIdentity`)
- add an application to the local identity notification queue (`AuthAddToLocalIdentityQueue`)
- remove an application from the local identity notification queue (`AuthRemoveFromLocalIdentityQueue`)

The Authentication Manager also provides functions that the PowerTalk Key Chain uses to

- set up the local identity (`AuthSetupLocalIdentity`)
- change the local identity (`AuthChangeLocalIdentity`)
- lock the local identify (`AuthLockLocalIdentity`)
- unlock the local identity (`AuthUnlockLocalIdentity`)
- remove the local identity (`AuthRemoveLocalIdentity`)

AuthGetLocalIdentity

Call the `AuthGetLocalIdentity` function to get the local identity.

```
pascal OSErr AuthGetLocalIdentity (AuthParamBlockPtr paramBlock,
                                   Boolean async);
```

`paramBlock`	A pointer to a parameter block.
`async`	A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
←	`theLocalIdentity`	`LocalIdentity`	The local identity

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`theLocalIdentity`
The local identity.

DESCRIPTION

You can call the `AuthGetLocalIdentity` function to obtain the local identity. If the local identity has not been set up, the `AuthGetLocalIdentity` function returns a `kOCEOCESetupRequired` result code. If the local identity is locked, the `AuthGetLocalIdentity` function returns a `kOCELocalAuthenticationFail` result code.

If your application is not a background application, you can call the `SDPPromptForID` function to prompt the user to unlock the local identity.

If your application runs only in the background, you can register with the Authentication Manager using the `AuthAddToLocalIdentityQueue` function. Then your application is notified when the local identity is created or unlocked.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0204

RESULT CODES

`noErr`	0	No error
`kOCELocalAuthenticationFail`	–1561	Local identity locked
`kOCESetupRequired`	–1633	Setup of local identity required

SEE ALSO

The `AuthGetLocalIdentity` function is used in an example in the section "Authentication Using ASDSP" on page 9-408.

The `SDPPromptForID` function is described in the chapter "Standard Catalog Package" in this book.

The `AuthAddToLocalIdentityQueue` function is discussed next.

AuthAddToLocalIdentityQueue

Call the `AuthAddToLocalIdentityQueue` function to add an application to the Authentication Manager's local identity notification queue.

```
pascal OSErr AuthAddToLocalIdentityQueue
               (AuthParamBlockPtr paramBlock, Boolean async);
```

`paramBlock`	A pointer to a parameter block.
`async`	A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`clientData`	`long`	For your use
→	`notifyProc`	`NotificationProc`	Notification function
→	`notifyFlags`	`AuthNotifications`	Notification flags
→	`appName`	`StringPtr`	Application name

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`clientdata`	A value for your use. The Authentication Manager passes the value in this field to your notification routine.
`notifyProc`	A pointer to your notification routine. You must provide a notification routine to be called by the notification queue.
`notifyFlags`	A flag byte that specifies when you want your notification routine to be called: when the local identity is about to be locked, when it is unlocked, when the user changes the name in the PowerTalk Key Chain, or for some combination of these events.
`appName`	A pointer to the name of your application.

DESCRIPTION

You call the `AuthAddToLocalIdentityQueue` function to add your notification routine to the Authentication Manager's notification queue.

You set the `notifyFlags` field to specify when you want your notification routine called. Possible values for this field are as follows:

```
enum {kNotifyLockBit, kNotifyUnlockBit, kNotifyNameChangeBit};
enum
   {kNotifyLockMask     = 1L << kNotifyLockBit,
   kNotifyUnlockMask    = 1L << kNotifyUnlockBit
   kNotifyNameChangeMask= 1L << kNotifyNameChangeBit
};
```

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0205

RESULT CODES

`noErr`	0	No error

SEE ALSO

For an example of the use of the `AuthAddToLocalIdentityQueue` function, see Listing 9-1 on page 9-411.

The notification routine is described on page 9-465.

AuthRemoveFromLocalIdentityQueue

Call the `AuthRemoveFromLocalIdentityQueue` function to remove your notification routine from the Authentication Manager's notification queue.

```
pascal OSErr AuthRemoveFromLocalIdentityQueue
                  (AuthParamBlockPtr paramBlock, Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`notifyProc`	`NotificationProc`	Notification function

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`notifyProc` The notification routine you provided when you called the `AuthAddToLocalIdentityQueue` function.

DESCRIPTION

You call the `AuthRemoveFromLocalIdentityQueue` function to remove your notification routine from the Authentication Manager's notification queue. The Authentication Manager informs the routines in the notification queue of changes in the state of the local identity access.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0206

RESULT CODES

`noErr`	0	No error

SEE ALSO

For an example of the use of the `AuthRemoveFromLocalIdentityQueue` function, see Listing 9-1 on page 9-411.

You use the `AuthAddToLocalIdentityQueue` function (page 9-426) to add a routine to the notification queue.

The notification procedure is described on page 9-465.

AuthSetupLocalIdentity

The `AuthSetupLocalIdentity` function sets up the user name and password for the local identity.

```
pascal OSErr AuthSetupLocalIdentity (AuthParamBlockPtr paramBlock,
                                     Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`userName`	`RStringPtr`	The user name
→	`password`	`RStringPtr`	The user password

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`userName`	The name of the principal user of the local computer.
`password`	The password to assign to the principal user of the local computer.

DESCRIPTION

You can use this function to set up the user name and password for the local identity. Normally, however, the user sets up a local identity by specifying a name and password in the PowerTalk Key Chain. You can call the `SDPPromptForID` function to prompt a user for a password to unlock the local identity when necessary.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0216

RESULT CODES

`noErr`	0	No error
`kOCELocalIdentitySetupExists`	–1562	Local identity setup exists, use `AuthChangeLocalIdentity` instead

SEE ALSO

You can use the `AuthGetLocalIdentity` function (page 9-424) to obtain a local identity once it has been set up.

Use the `SDPPromptForID` function, which is described in the chapter "Standard Catalog Package" in this book, to prompt the user for a name and password to unlock the local identity. This function also returns the local identity.

AuthChangeLocalIdentity

The `AuthChangeLocalIdentity` function changes the password for the local identity.

```
pascal OSErr AuthChangeLocalIdentity
                                        (AuthParamBlockPtr paramBlock,
                                         Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`userName`	`RStringPtr`	The user name
→	`password`	`RStringPtr`	The user password
→	`newPassword`	`RStringPtr`	The new user password

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`userName`	The name of the principal user of the local computer.
`password`	The current password for the principal user of the local computer.
`newPassword`	The new password you want to assign to the principal user of the local computer.

DESCRIPTION

You can use this function to change the password for the local identity from within your application. Normally, however, the user uses the PowerTalk Key Chain to change the password for the local identity.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0217

RESULT CODES

`noErr`	0	No error
`kOCELocalAuthenticationFail`	–1561	Local identity locked
`kOCEOCESetupRequired`	–1633	Setup of local identity required

SEE ALSO

You can use the `AuthSetupLocalIdentity` function (page 9-428) to set up a local identity.

AuthLockLocalIdentity

The `AuthLockLocalIdentity` function locks the local identity.

```
pascal OSErr AuthLockLocalIdentity (AuthParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`theLocalIdentity`	`LocalIdentity`	The local identity
←	`appName`	`StringPtr`	The name of the application that denied locking (if any)

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`theLocalIdentity`
The local identity.

`appName` The name of the application that denied locking, if the function fails and returns the `kOCEOperationDenied` result code. Allocate a pointer to a `Str31` data type for this parameter.

DESCRIPTION

To lock the local identity, a user can choose the Lock Key Chain command from the Special menu of the Finder or set the PowerTalk Setup control panel to lock the Key Chain after some specified period of inactivity. You can use the `AuthLockLocalIdentity` function to lock the local identity from within your application.

When you call the `AuthLockLocalIdentity` function, the Authentication Manager calls every routine in its notification queue to give it an opportunity to deny the lock operation. If any application denies the operation, the `AuthLockLocalIdentity` function returns the `kOCEOperationDenied` result code and the `appName` field points to the name of the application that denied the locking operation.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0215

RESULT CODES

`noErr`	0	No error
`kOCEOperationDenied`	–1568	Local identity operation denied

SEE ALSO

The notification queue is described in “Local Identity Status Notification” on page 9-405.

You use the `AuthAddToLocalIdentityQueue` function (page 9-426) to add a routine to the notification queue.

You can use the `AuthUnlockLocalIdentity` function (described next) to unlock a local identity.

AuthUnlockLocalIdentity

Call the `AuthUnlockLocalIdentity` function to unlock the local identity.

```
pascal OSErr AuthUnlockLocalIdentity
                                   (AuthParamBlockPtr paramBlock,
                                    Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`theLocalIdentity`	`LocalIdentity`	The local identity
→	`userName`	`RStringPtr`	The name of the user
→	`password`	`RStringPtr`	The user password

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`theLocalIdentity`	The local identity.
`userName`	The name of the principal user of the local computer.
`password`	The password for the principal user of the local computer.

DESCRIPTION

To unlock a local identity, the user can choose the Unlock Key Chain command from the Finder's Special menu. You can also call the `SDPPromptForID` function to prompt the user for a password and unlock the local identity. Alternatively, you can use the `AuthUnlockLocalIdentity` function to unlock the local identity from within your application. If the local identity does not exist, this function creates one.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0214

RESULT CODES

`noErr`	0	No error
`kOCEOCESetupRequired`	–1633	Setup of local identity required

SEE ALSO

The `AuthLockLocalIdentity` function is described on page 9-431.

The `AuthSetupLocalIdentity` function is described on page 9-428.

The `SDPPromptForID` function is described in the chapter "Standard Catalog Package" in this book.

AuthRemoveLocalIdentity

Call the `AuthRemoveLocalIdentity` function to remove the local identity.

```
pascal OSErr AuthRemoveLocalIdentity
                                    (AuthParamBlockPtr paramBlock,
                                     Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async`	A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`userName`	`RStringPtr`	The name of the user.
→	`password`	`RStringPtr`	The user password.

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`userName`	The name of the principal user of the local computer
`password`	The password for the principal user of the local computer

DESCRIPTION

Normally, a user cannot remove a local identity from a PowerTalk system without replacing it with a new local identity or reinstalling the PowerTalk system software. The user normally uses the Key Chain to change a local identity. You can use the `AuthRemoveLocalIdentity` function to remove the local identity, effectively rendering the Key Chain inoperable. The user then is prompted to set up a local identity the next time he or she attempts to use the PowerTalk system software.

IMPORTANT

Because removing the local identity disrupts the use of the PowerTalk system software on the user's computer, warn users before allowing them to remove a local identity. ▲

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0218

RESULT CODES

`noErr`	0	No error
`kOCELocalAuthenticationFail`	–1561	Local identity locked
`kOCEOCESetupRequired`	–1633	Setup of local identity required

SEE ALSO

To lock a local identity so that the user must enter the password before using PowerTalk, use the `AuthLockLocalIdentity` function (page 9-431).

Specific Identity Management

A specific identity is a shorthand representation for the name and key of an alternate user. See “Specific Identities” on page 9-405 for a further discussion.

The Authentication Manager provides the following specific identity management services:

- binding a new specific identity number to a user’s record ID and key (`AuthBindSpecificIdentity`)
- unbinding a specific identity number from a user’s record ID and key (`AuthUnbindSpecificIdentity`)
- using a specific identity to get a user’s record ID (`AuthGetSpecificIdentityInfo`)

AuthBindSpecificIdentity

Call the `AuthBindSpecificIdentity` function to bind an identity number to a specified authentication client’s record ID and key.

```
pascal OSErr AuthBindSpecificIdentity
                                (AuthParamBlockPtr paramBlock,
                                 Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
←	`userIdentity`	`AuthIdentity`	Binding identity
→	`userRecord`	`RecordIDPtr`	Entity’s record ID
→	`userKey`	`AuthKeyPtr`	Entity’s key

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`userIdentity`	The specific identity.
`userRecord`	A pointer to the record ID of the authentication client.
`userKey`	A pointer to the user or service key for the client.

DESCRIPTION

Call the `AuthBindSpecificIdentity` function to bind an identity to a record ID and key you provide. The Authentication Manager contacts the catalog containing the record identified by the `userRecord` field to verify the name and key. If the name is valid and the key is correct, the `AuthBindSpecificIdentity` function returns an identity.

You can use the identity returned by this function as an input to any AOCE function that requires an identity. The AOCE software uses the identity to check whether the authentication client has sufficient access privileges to do the operation requested.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0200

RESULT CODES

`noErr`	0	No error
`kOCENoKeyFound`	–1550	Client has no key
`kOCEWrongIdentityOrKey`	–1557	Incorrect key for client
`kOCEUnknownID`	–1567	Identity passed is not valid
`kOCENotLocal`	–1610	Internal AOCE error
`kOCETargetDirectoryInaccessible`	–1613	Catalog server not responding
`kOCENoSuchDNode`	–1615	The dNode was not found
`kOCEBadRecordID`	–1617	Name and type incorrect for creation ID
`kOCENoSuchRecord`	–1618	Record ID doesn’t exist
`kOCEStreamCreationErr`	–1625	An error occurred in creating the stream

SEE ALSO

The `AuthBindSpecificIdentity` function is used in an example in the section “Authentication Using ASDSP” on page 9-408.

You can use the `AuthPasswordToKey` function (page 9-417) to get a key from a password.

The `AuthUnbindSpecificIdentity` function is described next.

AuthUnbindSpecificIdentity

The `AuthUnbindSpecificIdentity` function unbinds an identity from a user's name and key.

```
pascal OSErr AuthUnbindSpecificIdentity
                  (AuthParamBlockPtr paramBlock, Boolean async);
```

`paramBlock`
: A pointer to a parameter block.

`async`
: A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`userIdentity`	`AuthIdentity`	Binding identity

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`userIdentity` The identity to be deleted.

DESCRIPTION

Call the `AuthUnbindSpecificIdentity` function to remove permanently an identity you no longer need; for example, when your application quits.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0201

RESULT CODES

`noErr`	0	No error
`kOCENotLocalIdentity`	–1565	You cannot unbind a local identity
`kOCEUnknownID`	–1567	Identity passed is not valid

SEE ALSO

The `AuthBindSpecificIdentity` function is described on page 9-435.

The `AuthGetSpecificIdentityInfo` function (described next) returns the record ID associated with a specific identity.

AuthGetSpecificIdentityInfo

Call the `AuthGetSpecificIdentityInfo` function to get the record ID (but not the user or service key) associated with the specified identity.

```
pascal OSErr AuthGetSpecificIdentityInfo
                                    (AuthParamBlockPtr paramBlock,
                                     Boolean async);
```

`paramBlock`	A pointer to a parameter block.
`async`	A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`userIdentity`	`AuthIdentity`	Identity
↔	`userRecord`	`RecordIDPtr`	Entity's record ID

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`userIdentity`	The identity whose record ID is desired.
`userRecord`	A pointer to the record ID structure for the record, in which the record ID is returned.

DESCRIPTION

Call the `AuthGetSpecificIdentityInfo` function to obtain the record ID associated with a particular identity.

The `userRecord` field must contain a pointer to a `recordID` structure of maximum size.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0203

RESULT CODES

`noErr`	0	No error
`kOCENotLocalIdentity`	–1565	You cannot unbind a local identity
`kOCEUnknownID`	–1567	Identity passed is not valid

SEE ALSO

The chapter "AOCE Utilities" in this book describes how to allocate space for a record ID.

The `AuthBindSpecificIdentity` function is described on page 9-435.

The `AuthUnbindSpecificIdentity` function is described on page 9-437.

Credentials Management

Credentials enable initiators and recipients to verify each other's identities. See "Credentials" on page 9-401 for more information. The Authentication Manager provides functions to

- get credentials from the server(`AuthGetCredentials`)
- obtain a proxy with which to get credentials (`AuthMakeProxy`)
- use a proxy to get credentials from the server (`AuthTradeProxyForCredentials`)

AuthGetCredentials

Call the `AuthGetCredentials` function to obtain credentials from the authentication server.

```
pascal OSErr AuthGetCredentials (AuthParamBlockPtr paramBlock,
                                 Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`userIdentity`	`AuthIdentity`	Initiator identity
→	`recipient`	`RecordIDPtr`	Record ID of recipient
↔	`sessionKey`	`AuthKeyPtr`	Session key
↔	`expiry`	`UTCTime`	Desired/actual times
↔	`credentialsLength`	`unsigned long`	Buffer size and credentials size
↔	`credentials`	`Ptr`	Credentials buffer

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`userIdentity`	The identity of the initiator.
`recipient`	A pointer to the record ID of the recipient.
`sessionKey`	A pointer to a buffer that you supply to the function. The function puts the session key into this buffer.
`expiry`	When you call the function, you use the `expiry` field to specify the time at which you want the credentials to expire. When the function completes, this field specifies the actual expiration time: your desired expiration time or the current time plus 8 hours, whichever is sooner.
`credentialsLength`	When you call the function, you use this field to specify the size of the buffer pointed to by the `credentials` field. A buffer three times the size of a packed record ID is usually sufficient for credentials. Use the `kPackedRecordIDMaxBytes` constant defined in the chapter "AOCE Utilities" in this book to determine the size of a packed record ID. When the function completes, this field indicates the actual amount of data written into the buffer.
`credentials`	A pointer to the buffer you provide to hold the returned credentials.

DESCRIPTION

Call the `AuthGetCredentials` function to get credentials to establish an authenticated connection with the named recipient. Any entity can request credentials for any other entity.

Your application should call the `AuthGetUTCTime` function before calling the `AuthGetCredentials` function because the expiration time you specify is based on universal coordinated time (UTC). You add the desired number of seconds to the current time returned by the `AuthGetUTCTime` function.

If the `AuthGetCredentials` function is successful, the buffer pointed to by the `credentials` field contains encrypted credentials and the `sessionKey` field contains the key to use during the challenge portion of the authentication process. The credentials returned by the server to the initiator are encrypted in the key of the recipient.

If the buffer you provide is not large enough to hold the credentials, the function returns the `kOCEMoreData` result code. You can increase the buffer size and call the function again.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$020B

RESULT CODES

`noErr`	0	No error
`kOCECredentialsExpired`	–1546	Desired expiration time has passed
`kOCERecipientKeyNotFound`	–1552	The recipient key was not found
`kOCEInitiatorKeyProblem`	–1558	No key, or initiator's key changed
`kOCEUnknownID`	–1567	Identity passed is not valid
`kOCENotLocal`	–1610	Internal AOCE error
`kOCETargetDirectoryInaccessible`	–1613	Catalog server not responding
`kOCENoSuchDNode`	–1615	The dNode was not found
`kOCEBadRecordID`	–1617	Name and type incorrect for creation ID
`kOCEMoreData`	–1623	Buffer was too small to hold all available data
`kOCEStreamCreationErr`	–1625	An error occurred in creating the stream

SEE ALSO

The authentication process is described in "Steps in the Authentication Process" beginning on page 9-401.

The `AuthGetCredentials` function is used in an example in the section "Authentication Using ASDSP" on page 9-408.

The `AuthGetUTCTime` function is discussed on page 9-449.

The `AuthDecryptCredentials` function is discussed on page 9-455.

AuthMakeProxy

Call the `AuthMakeProxy` function to create a proxy.

```
pascal OSErr AuthMakeProxy (AuthParamBlockPtr paramBlock,
                            Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`userIdentity`	`AuthIdentity`	Principal identity
→	`recipient`	`RecordIDPtr`	Recipient record ID
→	`firstValid`	`UTCTime`	Time proxy becomes valid
→	`expiry`	`UTCTime`	Time proxy expires
→	`authDataLength`	`unsigned long`	Must be 0
→	`authData`	`Ptr`	Must be `nil`
↔	`proxyLength`	`unsigned long`	Buffer size and proxy size
↔	`proxy`	`Ptr`	Proxy buffer
→	`intermediary`	`RecordIDPtr`	Intermediary record ID

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`userIdentity`	The identity of the user or service for which you are requesting the proxy.
`recipient`	A pointer to the record ID of the recipient.
`firstValid`	The time that the proxy is to become valid.
`expiry`	The last time at which you want the proxy to be valid
`authDataLength`	Reserved. Set this parameter to 0.
`authData`	Reserved. Set this parameter to `nil`.
`proxyLength`	The length of the buffer to which the `proxy` field points. A buffer twice the size of a packed record ID is usually sufficient for a proxy. Use the `kPackedRecordIDMaxBytes` constant defined in the chapter "AOCE Utilities" in this book to determine the size of a packed record ID. The function returns the actual length of the proxy in this parameter.
`proxy`	A pointer to the proxy buffer, in which the function returns the proxy.
`intermediary`	A pointer to the record ID of the intermediary that will use the proxy to obtain credentials in your behalf.

DESCRIPTION

Call the `AuthMakeProxy` function to create a proxy. A proxy is granted to an intermediary for use with a particular recipient during a specified time period only. The `AuthMakeProxy` function creates a proxy and returns it to you. You can then pass it to an intermediary to use on your behalf. The proxy is valid only until the expiration time you specify in the `expiry` field. To obtain credentials, the intermediary must call the `AuthTradeProxyForCredentials` function.

If the function returns a `kOCEMoreData` result code, you can call the `AuthMakeProxy` function again after increasing the buffer size.

SPECIAL CONSIDERATIONS

The Authentication Manager provides no mechanism for sending a proxy from an initiator to an intermediary. You must devise your own mechanism and protocol for this purpose.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0212

RESULT CODES

`noErr`	0	No error
`kOCEMoreData`	–1623	Buffer was too small to hold all available data

SEE ALSO

The `AuthMakeProxy` function is used in an example in the section “Authentication Using a Proxy” on page 9-410.

See “Proxies” on page 9-406 for a discussion of proxies and “Steps in the Authentication Process” beginning on page 9-401 for a description of the authentication process.

The `AuthTradeProxyForCredentials` function is described next.

AuthTradeProxyForCredentials

Call the `AuthTradeProxyForCredentials` function to trade a proxy for credentials.

```
pascal OSErr AuthTradeProxyForCredentials
                  (AuthParamBlockPtr paramBlock, Boolean async);
```

`paramBlock`	A pointer to a parameter block.
`async`	A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`userIdentity`	`AuthIdentity`	Intermediary identity
→	`recipient`	`RecordIDPtr`	Recipient name
↔	`sessionKey`	`AuthKeyPtr`	Session key
↔	`expiry`	`UTCTime`	Credentials expiration times
↔	`credentialsLength`	`unsigned long`	Buffer size and credentials size
↔	`credentials`	`Ptr`	Credentials buffer
→	`proxyLength`	`unsigned long`	Actual proxy size
→	`proxy`	`Ptr`	Proxy buffer
→	`principal`	`RecordIDPtr`	Record ID of principal

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`userIdentity`	The identity of the intermediary.
`recipient`	A pointer to the record ID of the recipient.
`sessionKey`	A pointer to the session key buffer that you supply. The function returns the session key in this buffer.
`expiry`	The desired expiration time for the credentials. The function returns the actual expiration time.
`credentialsLength`	As an input, the size of the buffer you are providing to hold the returned credentials. Use the `kPackedRecordIDMaxBytes` constant defined in the chapter "AOCE Utilities" in this book to determine the size needed. On return, this field holds the actual size of the credentials.
`credentials`	A pointer to the buffer in which the function places the encrypted credentials.
`proxyLength`	The size of the proxy.
`proxy`	A pointer to the buffer containing the proxy used to get the credentials.
`principal`	A pointer to the record ID of the user or service who created the proxy.

DESCRIPTION

Calling the `AuthTradeProxyForCredentials` function is very similar to calling the `AuthGetCredentials` function, except that the creator of the proxy first calls the `AuthMakeProxy` function to obtain a proxy and gives the proxy to an intermediary; then the intermediary calls the `AuthTradeProxyForCredentials` function for credentials. In the `principal` field, you specify the entity who made the proxy.

The expiration time of the credentials depends on the maximum lifetime permitted by the Authentication Manager, the period during which the proxy is valid, and the expiration time you request for the credentials. For example, assume that the proxy has an expiration time of 3:00 P.M. on a given day of a given month of a given year. Assume all other times in this example are for the same day, month, and year as the proxy expiration time. First, if it is 3:15 P.M. when the intermediary requests credentials, the Authentication Manager refuses the request because the proxy has expired. If, however, the intermediary requests credentials at 5:00 A.M., the credentials expire at 1:00 P.M. even though you requested a 3:00 P.M. expiration, because the server enforces a maximum lifetime for credentials of 8 hours. If you request credentials at any time between 7:01 A.M. and 2:59 P.M., the credentials expire at 3:00 P.M., because credentials must expire at or before the time specified by the proxy expiration time.

You can use the `AuthTradeProxyForCredentials` function to request credentials as many times as you wish during the lifetime of the proxy.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0213

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	No recipient, or invalid recipient dNode
`kOCEAccessRightsInsufficient`	–1542	Intermediary’s record ID does not appear in the proxy
`kOCEProxyImmature`	–1547	Proxy not yet valid
`kOCEProxyExpired`	–1548	Proxy has expired
`kOCEDisallowedRecipient`	–1549	Recipient record ID does not appear in proxy
`kOCERecipientKeyNotFound`	–1552	No key found
`kOCEAgentKeyNotFound`	–1553	Intermediary’s key not found
`kOCEInitiatorKeyProblem`	–1558	Can’t decipher instructions or the principal’s key was not found
`kOCEUnknownID`	–1567	Identity passed is not valid
`kOCENotLocal`	–1610	Internal AOCE error
`kOCETargetDirectoryInaccessible`	–1613	Catalog server not responding
`kOCENoSuchDNode`	–1615	The dNode was not found
`kOCEBadRecordID`	–1617	Name and type incorrect for creation ID of recipient or principal
`kOCEMoreData`	–1623	Buffer was too small to hold all available data
`kOCEStreamCreationErr`	–1625	An error occurred in creating the stream

SEE ALSO

The `AuthTradeProxyForCredentials` function is used in an example in the section “Authentication Using a Proxy” on page 9-410.

See “Proxies” on page 9-406 for a discussion of proxies and “Steps in the Authentication Process” beginning on page 9-401 for a description of the authentication process.

The `AuthGetCredentials` function is discussed on page 9-439.

The `AuthMakeProxy` function is discussed on page 9-441.

Creation ID Resolution

Creation IDs are unique identifiers for records. The are described in detail in the chapters “AOCE Utilities” and “Catalog Manager” in this book. The `AuthResolveCreationID` function returns the creation ID of a record with the name and type that you supply. If there are multiple records with the same name and type, then it returns the creation IDs of all of the records that match the name and type.

AuthResolveCreationID

Call the `AuthResolveCreationID` function to obtain all the dNode numbers and creation IDs for all the records that have a given name and type.

```
pascal OSErr AuthResolveCreationID (AuthParamBlockPtr paramBlock,
                                     Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`identity`	`AuthIdentity`	Identity; must be 0
→	`userRecord`	`RecordIDPtr`	A record ID
→	`bufferLength`	`unsigned long`	Buffer size
↔	`buffer`	`Ptr`	Data buffer
←	`totalMatches`	`unsigned long`	Number of matches found
←	`actualMatches`	`unsigned long`	Number of matches returned

See page 9-415 for descriptions of the `ioCompletion`, `ioResult`, and `identity` fields.

Field descriptions

`userRecord` A pointer to the record ID of the entity whose dNode number and creation ID are to be returned. You must specify the name and type for the entity. The `RLI` must include the dNode number or the

	pathname of the dNode in which you expect the user's record to be located. The `cid` field of the record ID must be set to `NULL` before the function is called.
`bufferLength`	The size of the buffer for holding dNode numbers and creation IDs.
`buffer`	A pointer to the buffer to hold dNode numbers and creation IDs.
`totalMatches`	Total number of matching names found in the server catalog.
`actualMatches`	Number of matches returned in the buffer. This number is determined by how many dNode numbers and creation IDs fit in the buffer.

DESCRIPTION

The creation ID is a unique identifier for a given record. If you don't know this identifier but know the record ID, you can determine the creation ID by calling the `AuthResolveCreationID` function. There may be several records with the same name and type. It is the responsibility of your user application to prompt users to choose the record desired from those provided by this function.

In most cases, you should search the Users and Groups folder, which has the dNode number 3, for the record. This folder normally contains the User record or an alias to the User record of every user with an account on the PowerShare server. If the Collaboration toolbox finds a record with the name and type you specify, it returns the dNode number and creation ID of that record. If it finds an alias to a record with the name and type you specify, it resolves the alias and returns the dNode number and creation ID of the original record.

You must set the creation ID of the record ID to `NULL` before calling the `AuthResolveCreationID` function. You do this by calling the `OCESetCreationIDToNull` function.

The server finds all records in the catalog whose name and type match those in the `userRecord` field. Depending on the number of matches, the following results are returned

- Exactly one match: the dNode number and creation ID are put in the buffer.
- More than one match if the buffer is large enough to hold all matches: The buffer contains the dNode numbers and creation IDs of all records with matching names and types. A `kOCEAmbiguousMatches` result code is returned.
- More than one match if the buffer is not large enough to hold all the matches: the `totalMatches` field contains the number of matches that were found in the server catalog. The `actualMatches` field contains how many of the dNode numbers and creation IDs fit in the buffer, and the buffer contains as many dNode numbers and creation IDs as fit, packed one after the other. A `kOCEMoreData` result code is returned.
- No matches: a `kOCENoSuchRecord` result code is returned.

When you have more than one match and the buffer is not large enough, you can call this function again using an appropriately sized buffer. The dNode numbers and creation IDs are loaded into the user buffer in an array the size of the `actualMatches` field.

SPECIAL CONSIDERATIONS

This function does not check access controls. You must pass a 0 in the `identity` field.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0202

RESULT CODES

`noErr`	0	No error
`kOCEParamErr`	–50	Invalid parameter
`kOCEUnknownID`	–1567	Identity passed is not valid
`kOCEAmbiguousMatches`	–1569	More than one match
`kOCENotLocal`	–1610	Internal AOCE error
`kOCETargetDirectoryInaccessible`	–1613	Catalog server not responding
`kOCENoSuchDNode`	–1615	The dNode was not found
`kOCENoSuchRecord`	–1618	No such record found with creation ID
`kOCEMoreData`	–1623	Buffer was too small to hold all available data
`kOCEStreamCreationErr`	–1625	An error occurred in creating the stream

SEE ALSO

The `OCESetCreationIDToNull` function is described in the chapter “AOCE Utilities” in this book.

Time Service

In a distributed system of many computers, you need a common time for communication. The Authentication Manager provides the universal coordinated time (UTC), also known as Greenwich Mean Time. You can use UTC to specify issue and expiration times for credentials and for other possible uses in your application. Call the `AuthGetUTCTime` function to get the current UTC.

AuthGetUTCTime

The `AuthGetUTCTime` function returns the current universal coordinated time (UTC) that is maintained by a catalog server.

```
pascal OSErr AuthGetUTCTime (AuthParamBlockPtr paramBlock,
                             Boolean async);
```

`paramBlock`	A pointer to a parameter block.
`async`	A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`pRLI`	`PackedRLIPtr`	Packed RLI of the node
←	`theUTCTime`	`UTCTime`	UTC seconds east of Greenwich
←	`theUTCOffset`	`UTCOffset`	Offset from UTC

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`pRLI`	Indicates which catalog to consult to determine the UTC. Time servers within a catalog communicate among themselves to determine their UTC. Servers in a different catalog might have a different value of UTC. If you pass a valid record location information structure (RLI), you get that catalog's version of UTC. If you pass `nil` as the value of the `pRLI` field, the Authentication Manager calculates the values of the `theUTCTime` and `theUTCOffset` fields according to the clock in the user's Macintosh computer and the settings in the Map control panel. Packed record location information structures are described in the chapter "AOCE Utilities" in this book.
`theUTCTime`	The function returns the current universal coordinated time (UTC) expressed as the number of seconds since 12:00 midnight, 1 January, 1904.
`theUTCOffset`	The function returns the difference between the user's Macintosh computer's clock and UTC at Greenwich, England, expressed as the number of seconds. A negative number indicates that the user's computer is west of Greenwich according to the setting in the Map control panel.

DESCRIPTION

Call the `AuthGetUTCTime` function to obtain the current UTC. When you provide a valid RLI for a catalog, the function determines the UTC from the catalog server and local time from the settings in the Map control panel. The function returns the current UTC seconds since 1/1/1904 along with the offset from UTC in seconds of the local time, based on the distance of the local computer from Greenwich, England. Other Authentication Manager functions require input parameters based on UTC.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$021A

RESULT CODES

`noErr`	0	No error
`kOCEUnknownID`	–1567	Identity passed is not valid
`kOCENotLocal`	–1610	Internal AOCE error
`kOCETargetDirectoryInaccessible`	–1613	Catalog server not responding
`kOCENoSuchDNode`	–1615	The dNode was not found
`kOCEStreamCreationErr`	–1625	An error occurred in creating the stream

SEE ALSO

The `AuthGetUTCTime` function is used in an example in the section “Authentication Using ASDSP” on page 9-408.

Non-ASDSP Authentication Utilities

After obtaining credentials using the `AuthGetCredentials` function or the `AuthTradeProxyForCredentials` function, if you are not using the ASDSP transport mechanism, you can call functions to help you complete the challenge phase of authentication directly. This process for authenticating users is described in “Authentication for Non-ASDSP Users” beginning on page 9-409.

The Authentication Manager provides functions to

- make a challenge (`AuthMakeChallenge`)
- generate a reply to the challenge and a counterchallenge (`AuthMakeReply`)
- verify the reply and reply to the counterchallenge (`AuthVerifyReply`)
- extract information from the credentials (`AuthDecryptCredentials`)

AuthMakeChallenge

Call the `AuthMakeChallenge` function to generate a challenge, encrypted in the session key.

```
pascal OSErr AuthMakeChallenge (AuthParamBlockPtr paramBlock,
                                Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`key`	`AuthKeyPtr`	Session key
↔	`challenge`	`Ptr`	Challenge buffer
→	`challengeBufferLength`	`unsigned long`	Challenge buffer size
←	`challengeLength`	`unsigned long`	Challenge length

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`key` A pointer to the session key.

`challenge` A pointer to a buffer you provide in which to put the encrypted challenge.

`challengeBufferLength`
The size of the challenge buffer. The buffer must be at least 8 bytes in size.

`challengeLength`
The length of the encrypted challenge.

DESCRIPTION

An application that does not use ASDSP as the transport mechanism calls the `AuthMakeChallenge` function when it begins the process of setting up a new authenticated connection. Prior to calling this function, the application must obtain credentials from the authentication server using the `AuthGetCredentials` function or the `AuthTradeProxyForCredentials` function.

The `AuthMakeChallenge` function generates a token (a random number as described in the section "Steps in the Authentication Process" beginning on page 9-401), and encrypts it with the session key to create a challenge. You must then send the challenge to the recipient. Only initiators call this function.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$020F

RESULT CODES

`noErr`	0	No error
`kOCELengthError`	–1637	The supplied buffer was too small

SEE ALSO

The `AuthMakeChallenge` function is used in an example in the section "Authentication for Non-ASDSP Users" on page 9-409.

The `AuthGetCredentials` function is described on page 9-439.

The `AuthTradeProxyForCredentials` function is described on page 9-443.

The recipient uses the `AuthMakeReply` function, described next, to reply to the challenge.

AuthMakeReply

The `AuthMakeReply` function uses the token from an initial challenge to generate another token to be used as a challenge reply and also makes a counterchallenge.

```
pascal OSErr AuthMakeReply (AuthParamBlockPtr paramBlock,
                           Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`key`	`AuthKeyPtr`	Session key
→	`challenge`	`Ptr`	Challenge
↔	`reply`	`Ptr`	Reply buffer pointer
→	`replyBufferLength`	`unsigned long`	Reply buffer length
→	`challengeLength`	`unsigned long`	Challenge length
←	`replyLength`	`unsigned long`	Length of reply

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`key`	The session key.
`challenge`	The challenge that was received from the initiator.
`reply`	A pointer to the buffer you supply into which the function puts the reply and the counterchallenge.
`replyBufferLength`	The length of the challenge reply buffer.
`challengeLength`	The length of the challenge.
`replyLength`	The length of the reply.

DESCRIPTION

The `AuthMakeReply` function decrypts a challenge created by the `AuthMakeChallenge` function, increments by 1 the number contained in the challenge, and then encrypts that new number in the session key. The result is the challenge reply. If you are a recipient, you call the `AuthMakeReply` function after you use the `AuthDecryptCredentials` function to decrypt the credentials—which are encrypted in your client key—to obtain the session key.

The `AuthMakeReply` function places in your buffer the reply to the challenge plus a counterchallenge. After you send the reply and counterchallenge to the initiator, the initiator calls the `AuthVerifyReply` function to verify the reply, thus continuing the challenge phase for authenticating a connection. The `AuthMakeReply` function is called only by recipients.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0210

RESULT CODES

`noErr`	0	No error
`kOCELengthError`	–1637	The supplied buffer was too small

SEE ALSO

The `AuthMakeReply` function is used in an example in the section “Authentication for Non-ASDSP Users” on page 9-409.

Use the `AuthDecryptCredentials` function (page 9-455) to extract the session key from the encrypted credentials.

The `AuthMakeChallenge` function is described on page 9-451. The `AuthVerifyReply` function is discussed next.

AuthVerifyReply

The `AuthVerifyReply` function verifies a challenge reply and makes a reply to the counterchallenge.

```
pascal OSErr AuthVerifyReply (AuthParamBlockPtr paramBlock,
                              Boolean async);
```

`paramBlock`	A pointer to a parameter block.
`async`	A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`key`	`AuthKeyPtr`	Session key
→	`challenge`	`Ptr`	Challenge
↔	`reply`	`Ptr`	Reply buffer
→	`challengeLength`	`unsigned long`	Length of challenge
↔	`replyLength`	`unsigned long`	Length of reply

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`key`	A pointer to the session key.
`challenge`	A pointer to the challenge you sent last.
`reply`	A pointer to a buffer containing the reply returned by the other end of the connection.
`challengeLength`	The length of the challenge.
`replyLength`	The length of the reply.

DESCRIPTION

Call the `AuthVerifyReply` function to verify a challenge reply and to make a reply to the counterchallenge during the challenge phase of setting up a secure connection. Both the initiator and the recipient call this function to verify the challenge replies they receive.

This function returns the result code `noErr` if the reply, after decryption, equals the challenge sent plus 1. A value of `kOCEAuthenticationTrouble` is returned by the `AuthVerifyReply` function if the reply cannot be verified. In that case, authentication has failed, and you should either terminate communication with the other party or continue communication with the understanding that the other party is not an authenticated entity.

After calling this function, the initiator should send the recipient the contents of the buffer pointed to by the `reply` field.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$0211

RESULT CODES

`noErr`	0	No error
`kOCEAuthenticationTrouble`	–1571	Reply incorrect for the challenge sent

SEE ALSO

The `AuthVerifyReply` function is used in an example in the section “Authentication for Non-ASDSP Users” on page 9-409.

The `AuthMakeReply` function is described on page 9-452.

AuthDecryptCredentials

The `AuthDecryptCredentials` function decrypts credentials, extracting the session key, a pointer to the initiator’s record ID, and the issue and expiration times for the credentials. Additionally, if an intermediary used a proxy to generate the credentials, the function returns a pointer to the record ID for the intermediary.

```
pascal OSErr AuthDecryptCredentials (AuthParamBlockPtr paramBlock,
                                     Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`userIdentity`	`AuthIdentity`	Recipient's identity
↔	`initiatorRecord`	`RecordIDPtr`	Initiator's record ID
←	`sessionKey`	`AuthKeyPtr`	Session key
←	`expiry`	`UTCTime`	Credentials expiry time
←	`issueTime`	`UTCTime`	Credentials issue time
→	`credentialsLength`	`unsigned long`	Actual credentials size
→	`credentials`	`Ptr`	Credentials to be decrypted
←	`hasIntermediary`	`Boolean`	Intermediary found flag
↔	`intermediary`	`RecordIDPtr`	Intermediary who called

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`userIdentity`	The identity of the recipient wanting to decrypt credentials. The Authentication Manager gets your client key from your user record.
`initiatorRecord`	The record ID of the entity initiating the challenge process. If you pass a local identity in the `userIdentity` field, you must pass in the `initiatorRecord` field a record ID containing a record location information structure (`RLI` struct) that specifies the catalog of the recipient. The function returns the record ID of the initiator in this field.
`sessionKey`	The session key.
`expiry`	The expiration time for the credentials.
`issueTime`	The credentials issue time.
`credentialsLength`	The size of the credentials.
`credentials`	A pointer to the buffer holding the credentials to be decrypted.
`hasIntermediary`	A Boolean value indicating whether the credentials were sent by an intermediary. If `true`, these credentials were obtained via a proxy by calling the `AuthTradeProxyForCredentials` function.
`intermediary`	A pointer to the record ID of an intermediary, if any. You must allocate the record ID structure when you call the function. If you specify `nil` for this pointer, the function does not return the intermediary's record ID.

DESCRIPTION

When you are not using ASDSP as the transport mechanism, a recipient can use the `AuthDecryptCredentials` function to decrypt credentials received during a challenge. ASDSP decrypts credentials for its users, so you do not need to call the `AuthDecryptCredentials` function if you are using ASDSP.

Because the credentials are encrypted in the client key of the intended recipient, the function fails (with the result code `kOCEUnsupportedCredentialsVersion`) if you were not the intended recipient.

The `sessionKey` field is also given to the user or service requesting the decrypted credentials so that communicating users or services can share a key temporarily. You use this information to make encrypted challenge and challenge reply messages to complete the authentication process.

It is up to the user or service to refuse service if the credentials are premature or have expired.

If the function completes successfully, the `initiatorRecord`, `sessionKey`, `expiry`, `issueTime`, and `intermediary` fields contain plain text information extracted from the credentials.

SPECIAL CONSIDERATIONS

The recipient and initiator must be using the same PowerShare catalog for this function to succeed.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$020C

RESULT CODES

`noErr`	0	No error
`kOCEUnsupportedCredentialsVersion`	–1543	Problem reading the credentials

SEE ALSO

The `AuthDecryptCredentials` function is used in an example in the section "Authentication for Non-ASDSP Users" on page 9-409.

The `AuthGetCredentials` function is discussed on page 9-439.

The `AuthTradeProxyForCredentials` function is described on page 9-443.

PowerTalk Setup Catalog Management

The PowerTalk Setup catalog is a special personal catalog that contains information about the catalogs and electronic mail systems that are available to the principal user of the computer (see "The PowerTalk Setup Catalog" on page 9-405). Only CSAM and personal-MSAM template developers need to use the functions described in this section. If you are writing an application, you do not need to use these functions. See the chapter

"Service Access Module Setup" in *Inside Macintosh: Service Access Modules* for a complete description of setup templates and the PowerTalk Setup catalog.

The Authentication Manager provides functions associated with the PowerTalk Setup catalog to

- get the record ID and native name for a catalog in the PowerTalk Setup catalog (`OCESetupGetDirectoryInfo`)
- install catalogs and their passwords in the PowerTalk Setup catalog (`OCESetupAddDirectoryInfo`)
- change the password used to access a catalog in the PowerTalk Setup catalog (`OCESetupChangeDirectoryInfo`)
- remove a catalog from the PowerTalk Setup catalog (`OCESetupRemoveDirectoryInfo`)

OCESetupGetDirectoryInfo

Call the `OCESetupGetDirectoryInfo` function to get the record ID and native name of a specified catalog.

```
pascal OSErr OCESetupGetDirectoryInfo
                  (AuthParamBlockPtr paramBlock, Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`directoryName`	`DirectoryNamePtr`	Catalog name
→	`discriminator`	`DirDiscriminator`	Catalog discriminator
↔	`recordID`	`RecordIDPtr`	Catalog record ID
↔	`nativeName`	`RStringPtr`	User's name
↔	`password`	`RStringPtr`	Password

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`directoryName` A pointer to the catalog name.

`discriminator` A value that differentiates two catalogs with the same name. It is part of the `RLI` structure.

`recordID` A pointer to a record ID structure into which the function places the record ID of the catalog.

`nativeName`	A pointer to an `RString` structure into which the function places the native name. Allocate a buffer large enough to hold an `RString64` structure to hold this name.
`password`	For non-PowerShare catalogs, a pointer to an `RString` structure into which the function places the user or service password. Allocate a buffer large enough to hold an `RString64` structure to hold this password. This field is undefined for PowerShare catalogs.

DESCRIPTION

Call the `OCESetupGetDirectoryInfo` function to obtain the native name and record ID for a particular catalog installed in the PowerTalk Setup catalog. You specify the catalog name and discriminator. The *native name* is generally the user's name or account name in the external catalog, if it is different from the name of the user's User record. The CSAM or MSAM developer specifies this native name when installing the SAM in the Setup catalog.

The Collaboration toolbox returns the password only for non-PowerShare catalogs. An MSAM or CSAM can use this function to obtain from the Setup catalog the password required by the external system the SAM supports.

You must provide the buffers for the record ID, native name, and password that are returned.

SPECIAL CONSIDERATIONS

The local ID must be unlocked before you call this function.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	**Selector**
`_oceTBDispatch`	$020E

RESULT CODES

`noErr`	0	No error
`kOCELocalAuthenticationFail`	–1561	Local identity locked
`kOCEDirectoryIdentitySetupDoesNotExist`	–1564	Specific catalog has not been set up

SEE ALSO

See "The PowerTalk Setup Catalog" on page 9-405 for a description of the PowerTalk Setup catalog. See the chapter "Service Access Module Setup" in *Inside Macintosh: Service Access Modules* for a complete description of setup templates.

Record IDs and `RLI` structures are described in the chapter "AOCE Utilities" in this book.

The chapter "AOCE Utilities" in this book shows sample code that allocates space for a record ID.

OCESetupAddDirectoryInfo

Call the `OCESetupAddDirectoryInfo` function to add a catalog and its associated password to the PowerTalk Setup catalog.

```
pascal OSErr OCESetupAddDirectoryInfo
               (AuthParamBlockPtr paramBlock, Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`directoryRecordCID`	`CreationID`	Creation ID of catalog record
→	`recordID`	`RecordIDPtr`	Record ID for catalog
→	`password`	`RStringPtr`	Password

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`directoryRecordCID`
The creation ID of the Combined record or Catalog record in the Setup catalog. You can use the `kDETcmdGetDSSpec` template callback function to determine the creation ID of the Combined record or Catalog record.

`recordID` A pointer to the record ID specifying the user for the catalog.

`password` A pointer to the password associated with the record ID in the catalog.

DESCRIPTION

Only a setup template for a service access module (SAM) calls the `OCESetupAddDirectoryInfo` function. Before calling the `OCESetupAddDirectoryInfo` function, be sure the local identity is unlocked.

The `RLI` data structure within the user's record ID must contain the catalog name to be added to the Combined record or Catalog record.

The AOCE software encrypts the password before putting it in the PowerTalk Setup catalog.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$0219

RESULT CODES

`noErr`	0	No error
`kOCELocalAuthenticationFail`	–1561	Local identity locked
`kOCEDirectoryIdentitySetupExists`	–1563	Identity has already been set up
`kOCEDirectoryNotFoundErr`	–1630	Catalog was not found in the list

SEE ALSO

Creation IDs and the `RLI` structure are discussed in the chapter "AOCE Utilities" in this book.

The `kDETcmdGetDSSpec` template callback function is described in the chapter "AOCE Templates" in this book.

Setup templates and the procedure for adding a SAM to the Setup catalog are described in the chapter "Service Access Module Setup" in *Inside Macintosh: Service Access Modules.*

OCESetupChangeDirectoryInfo

Call the `OCESetupChangeDirectoryInfo` function to change the record ID and password for an existing catalog in the PowerTalk Setup catalog. The Authentication Manager verifies the current catalog password before changing it to the new password.

```
pascal OSErr OCESetupChangeDirectoryInfo
                    (AuthParamBlockPtr paramBlock, Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`directoryRecordCID`	`CreationID`	Catalog creation ID
→	`recordID`	`RecordIDPtr`	User's record ID
→	`password`	`RStringPtr`	Password
→	`newPassword`	`RStringPtr`	New password

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`directoryRecordCID`	The creation ID of the Combined record or Catalog record in the Setup catalog. You can use the `kDETcmdGetDSSpec` template callback function to determine the creation ID of the Combined record or Catalog record.
`recordID`	A pointer to the new record ID for the user. If you don't want to change the record ID, specify the old record ID.
`password`	A pointer to the current password associated with the record ID.
`newPassword`	A pointer to the new password to be associated with the record ID. If you don't want to change the password, repeat the old password in this field.

DESCRIPTION

Only a setup template for a SAM calls this function. Before calling the `OCESetupChangeDirectoryInfo` function, be sure the local identity is unlocked.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$021B

RESULT CODES

`noErr`	0	No error
`kOCELocalAuthenticationFail`	–1561	Local identity locked
`kOCEDirectoryNotFoundErr`	–1630	Catalog was not found in the list

SEE ALSO

Creation IDs and record IDs are discussed in the chapter "AOCE Utilities" in this book.

The `kDETcmdGetDSSpec` template callback function is described in the chapter "AOCE Templates" in this book.

OCESetupRemoveDirectoryInfo

Call the `OCESetupRemoveDirectoryInfo` function to remove a catalog from the PowerTalk Setup catalog.

```
pascal OSErr OCESetupRemoveDirectoryInfo
                    (AuthParamBlockPtr paramBlock, Boolean async);
```

`paramBlock`
A pointer to a parameter block.

`async` A value that specifies whether the function is to be executed asynchronously. Set this parameter to `true` if you want the function to be executed asynchronously.

Parameter block

→	`ioCompletion`	`ProcPtr`	Your completion routine
←	`ioResult`	`OSErr`	Result code
→	`directoryRecordCID`	`CreationID`	Catalog creation ID

See page 9-415 for descriptions of the `ioCompletion` and `ioResult` fields.

Field descriptions

`directoryRecordCID`
The creation ID for the Catalog record or Combined record associated with the catalog to be removed from the PowerTalk Setup catalog. You can use the `kDETcmdGetDSSpec` template callback function to determine the creation ID of the Combined record or Catalog record.

DESCRIPTION

Only a setup template for a SAM can call the `OCESetupRemoveDirectoryInfo` function. This function removes from the Catalog or Combined record in the PowerTalk Setup catalog the attributes that were added by the `OCESetupAddDirectoryInfo` function.

Before calling the `OCESetupRemoveDirectoryInfo` function, be sure the local identity is unlocked.

ASSEMBLY-LANGUAGE INFORMATION

Trap macro	Selector
`_oceTBDispatch`	$020D

RESULT CODES

`noErr`	0	No error
`kOCENoSuchRecord`	–1618	No such record

SEE ALSO

The `kDETcmdGetDSSpec` template callback function is described in the chapter “AOCE Templates” in this book.

Use the `OCESetupAddDirectoryInfo` function (page 9-460) to add a catalog and its associated password to the PowerTalk Setup catalog.

Use the `OCESetupChangeDirectoryInfo` function (page 9-461) to change the record ID and password for an existing catalog in the PowerTalk Setup catalog.

Application-Defined Functions

This section describes the completion routine required for asynchronous use of authentication functions and the notification routine that you provide to Authentication Manager functions that use a notification queue.

MyCompletion

An Authentication Manager completion routine has the following syntax:

```
void MyCompletion (Ptr paramBlk);
```

`paramBlk`	A pointer to the parameter block that you provided when you called the Authentication Manager function.

DESCRIPTION

When you execute an Authentication Manager function asynchronously (by setting its `async` parameter to `true`) you can specify a completion routine by passing the routine's address in the `ioCompletion` field of the parameter block. A function called asynchronously returns control to your application with the result code `noErr` as soon as the function is placed in the execution queue. This result code does not indicate that the function has successfully completed but indicates only that the function was successfully placed in the queue. To determine when the function is actually completed, you can inspect the `ioResult` field of the parameter block. This field is set to 1 when the function is called and set to the actual result code when the function is completed. If you specify a completion routine, it is executed after the result code is placed in the `ioResult` field.

SPECIAL CONSIDERATIONS

Because a completion routine may be executed at interrupt time, it should not allocate, move, or purge memory (either directly or indirectly) and should not depend on the validity of handles to unlocked blocks.

When the Authentication Manager calls your completion routine, it sets the A5 register to the value it contained when you called the function that set up the completion routine.

ASSEMBLY-LANGUAGE INFORMATION

When your completion routine is called, register A0 contains a pointer to the parameter block of the function called, and register D0 contains the result code. The value in register D0 is always identical to the value in the `ioResult` field of the parameter block.

A completion routine must preserve all registers other than A0, A1, and D0–D2.

MyNotificationProc

The `MyNotificationProc` function is a notification routine you must provide when you use the notification queue.

```
pascal Boolean MyNotificationProc (long clientData,
                       AuthLocalIdentityOp callValue,
                       AuthLocalIdentityLockAction actionValue,
                       LocalIdentity identity);
```

`clientData`
The value that you provided in the `clientData` field of the parameter block that you passed to the `AuthAddToLocalIdentityQueue` function. This field provides a way for you to pass a parameter to your notification routine.

`callValue` When the Authentication Manager calls your notification routine, it sets this parameter to `kAuthLockLocalIdentityOp` to indicate a lock operation, `kAuthUnlockLocalIdentityOp` to indicate an unlock operation, or to `kAuthLocalIdentityNameChangeOp` to indicate a name change. In the case of a lock operation, you must also check the value of the `actionValue` parameter.

`actionValue`
When the Authentication Manager calls your notification routine with the `kAuthLockLocalIdentityOp` value in the `callValue` parameter, it sets the `actionValue` parameter to either `kAuthLockPending`, indicating a lock is pending, or to `kAuthLockWillBeDone` when a lock is about to be done.

`identity` The local identity.

DESCRIPTION

The AOCE toolbox calls the notification procedure you provide each time the local identity access to a user’s computer is locked or unlocked, or when the user changes in the name in the Key Chain, so that the applications in the notification queue can be informed of changes in the access to catalogs listed in the PowerTalk Setup catalog.

Applications registered in the notification queue are notified when a user locks his or her local identity because he or she is leaving a computer unattended, and again when the user returns and provides his or her password to the system.

When it plans to lock local identity access, the Authentication Manager notifies all applications installed in the notification queue. To do so, the Authentication Manager passes the value `kAuthLockPending` in the `actionValue` parameter. Your notification procedure can return `true` to deny permission to lock the local identity. If none of the applications in the queue refuse the lock operation, the Collaboration toolbox passes the value `kAuthLockWillBeDone` to notify the applications that the lock is imminent.

You should deny locking only if you are performing some operation that would be seriously disrupted if the lock function succeeded.

The Authentication Manager handles the buffers associated with pointers that it passes to a notification procedure. You must copy the data in these buffers if you want to refer to it after your notification procedure completes execution.

SPECIAL CONSIDERATIONS

This routine should not allocate, move, or purge memory (either directly or indirectly). Like completion routines, your notification procedure should not call the `WaitNextEvent`, `EventAvail`, `OSEventAvail`, or `SystemTask` routines or any routine that might call those functions.

SEE ALSO

For an example of the use of the `MyNotificationProc` function, see Listing 9-1 on page 9-411.

See “Locking and Unlocking Local Identities” on page 9-404 for more information about locking and unlocking users’ computers.

See “The PowerTalk Setup Catalog” on page 9-405 for more information about the PowerTalk Setup catalog.

The `AuthAddToLocalIdentityQueue` function is discussed on page 9-426.

The `AuthRemoveFromLocalIdentityQueue` function is discussed on page 9-427.

Summary of the Authentication Manager

C Summary

Constants and Data Types

```
enum {
      /* values for key sizes */
      kRC4KeySizeInBytes               = 8,     /* size of an RC4 key */
      kRefNumUnknown                   = 0      /* dsRefNum specifier */
};

enum {
      /* values of AuthLocalIdentityOp for notification routine */
      kAuthLockLocalIdentityOp         = 1,
      kAuthUnlockLocalIdentityOp       = 2,
      kAuthLocalIdentityNameChangeOp   = 3
};

enum {
      /* values of AuthLocalIdentityLockAction for notification routine */
      kAuthLockPending                 = 1,
      kAuthLockWillBeDone              = 2
};

/* values of notifyFlags field of AuthAddToLocalIdentityQueue function*/
enum {kNotifyLockBit, kNotifyUnlockBit, kNotifyNameChangeBit};
enum {
      kNotifyLockMask                  = 1L << kNotifyLockBit,
      kNotifyUnlockMask                = 1L << kNotifyUnlockBit
      kNotifyNameChangeMask            = 1L << kNotifyNameChangeBit
};
```

Identity Declarations

```
typedef unsigned long AuthIdentity;                /* identity */
typedef AuthIdentity LocalIdentity;                /* local identity */
typedef unsigned long AuthLocalIdentityOp;
typedef unsigned long AuthLocalIdentityLockAction;
typedef unsigned long AuthNotifications;
```

Key Structures

```
struct DESKey { /* A DES key is 8 bytes of data */
   unsigned long a;
   unsigned long b;
};
typedef struct DESKey DESKey;
typedef Byte RC4Key[kRC4KeySizeInBytes];
typedef unsigned long AuthKeyType;
struct AuthKey {                          /* key type followed by its data */
   AuthKeyType keyType;
   union {
            DESKey des;
            RC4Key rc4;
         }u;
};
typedef struct Authkey AuthKey;
typedef AuthKey *AuthKeyPtr;
```

Parameter Block Header

```
#define AuthDirParamHeader
   Ptr             qLink;          /* reserved */
   long            reserved1;      /* reserved */
   long            reserved2;      /* reserved */
   ProcPtr         ioCompletion;   /* your completion function */
   OSErr           ioResult;       /* result code */
   unsigned long   saveA5;         /* reserved */
   short           reqCode;        /* reserved */
   long            reserved[2];    /* reserved */
   AddrBlock       serverHint;     /* PowerShare server’s AppleTalk address */
   short           dsRefNum;       /* reserved */
   unsigned long   callID;         /* reserved */
   AuthIdentity    identity;       /* initiator’s authentication identity */
   long            gReserved1;     /* reserved */
   long           gReserved2;    /* reserved */
   long           gReserved3;    /* reserved */
   long           clientData;    /* you define this field */
```

Parameter Blocks

```
struct AuthPasswordToKeyPB {
   AuthDirParamHeader
   RecordIDPtr          userRecord; /* User record */
   AuthKeyPtr           key;
   RStringPtr           password;   /* pointer to the new password string */
};
typedef struct AuthPasswordToKeyPB AuthPasswordToKeyPB;

struct AuthAddKeyPB {
   AuthDirParamHeader
   RecordIDPtr          userRecord; /* User record */
   AuthKeyPtr           userKey;    /* AOCE key for the user */
   RStringPtr           password;   /* pointer to password string */
};
typedef struct AuthAddKeyPB AuthAddKeyPB;

struct AuthChangeKeyPB {
   AuthDirParamHeader
   RecordIDPtr          userRecord; /* User record */
   AuthKeyPtr           userKey;    /* new AOCE key for the user */
   RStringPtr           password;   /* pointer to the new password string */
};
typedef struct AuthChangeKeyPB AuthChangeKeyPB;

struct AuthDeleteKeyPB {
   AuthDirParamHeader
   RecordIDPtr          userRecord; /* User record */
};
typedef struct AuthDeleteKeyPB AuthDeleteKeyPB;

struct AuthGetLocalIdentityPB {
   AuthDirParamHeader
   LocalIdentity        theLocalIdentity; /* local identity */
};
typedef struct AuthGetLocalIdentityPB AuthGetLocalIdentityPB;

struct AuthAddToLocalIdentityQueuePB {
   AuthDirParamHeader
   NotificationProc     notifyProc;    /* notification procedure */
   AuthNotifications    notifyFlags;   /* notifyFlags */
   StringPtr            appName;       /* name of application to be
                                          returned in Delete/Stop */
};
typedef struct AuthAddToLocalIdentityQueuePB AuthAddToLocalIdentityQueuePB;

struct AuthRemoveFromLocalIdentityQueuePB {
   AuthDirParamHeader
   NotificationProc     notifyProc;    /* notification procedure */
};
typedef struct AuthRemoveFromLocalIdentityQueuePB
AuthRemoveFromLocalIdentityQueuePB;

struct AuthSetupLocalIdentityPB {
   AuthDirParamHeader
   long                 aReserved;
   RStringPtr           userName;      /* user name */
   RStringPtr           password;      /* user password */
};
typedef struct AuthSetupLocalIdentityPB AuthSetupLocalIdentityPB;

struct AuthChangeLocalIdentityPB {
   AuthDirParamHeader
   long                 aReserved;
   RStringPtr           userName;      /* user name */
   RStringPtr           password;      /* current password */
   RStringPtr           newPassword;   /* new password */
};
typedef struct AuthChangeLocalIdentityPB AuthChangeLocalIdentityPB;

struct AuthLockLocalIdentityPB {
   AuthDirParamHeader
   LocalIdentity        theLocalIdentity; /* local identity */
   StringPtr            name;             /* name of the app that
                                             denied delete */
};
typedef struct AuthLockLocalIdentityPB AuthLockLocalIdentityPB;

struct AuthUnlockLocalIdentityPB {
   AuthDirParamHeader
   LocalIdentity        theLocalIdentity; /* local identity */
   RStringPtr           userName;         /* user name */
   RStringPtr           password;         /* user password */
};
typedef struct AuthUnlockLocalIdentityPB AuthUnlockLocalIdentityPB;

struct AuthRemoveLocalIdentityPB {
   AuthDirParamHeader
   long                 aReserved;
   RStringPtr           userName;         /* user name */
   RStringPtr           password;         /* current password */
};
typedef struct AuthRemoveLocalIdentityPB AuthRemoveLocalIdentityPB;

struct AuthBindSpecificIdentityPB {
   AuthDirParamHeader
   AuthIdentity         userIdentity;     /* binding identity */
   RecordIDPtr          userRecord;       /* User record */
   AuthKeyPtr           userKey;          /* AOCE key for the user */
};
typedef struct AuthBindSpecificIdentityPB AuthBindSpecificIdentityPB;

struct AuthUnbindSpecificIdentityPB {
   AuthDirParamHeader
   AuthIdentity         userIdentity;     /* identity to be deleted */
};
typedef struct AuthUnbindSpecificIdentityPB AuthUnbindSpecificIdentityPB;

struct AuthGetSpecificIdentityInfoPB {
   AuthDirParamHeader
   AuthIdentity         userIdentity;     /* identity of initiator */
   RecordIDPtr          userRecord;       /* User record */
};
typedef struct AuthGetSpecificIdentityInfoPB AuthGetSpecificIdentityInfoPB;

struct AuthGetCredentialsPB {
   AuthDirParamHeader
   AuthIdentity         userIdentity;     /* identity of initiator */
   RecordIDPtr          recipient;        /* AOCE name of recipient */
   AuthKeyPtr           sessionKey;       /* session key */
   UTCTime              expiry;           /* desired/actual expiration */
   unsigned long        credentialsLength;/* max/actual credentials size */
   Ptr                  credentials;      /* buffer where credentials
                                             are returned */
};
typedef struct AuthGetCredentialsPB AuthGetCredentialsPB;
```

```
struct AuthMakeProxyPB {
   AuthDirParamHeader
   AuthIdentity         userIdentity;  /* identity of principal */
   RecordIDPtr          recipient;     /* AOCE name of recipient */
   UTCTime              firstValid;    /* time at which proxy
                                           becomes valid */
   UTCTime              expiry;        /* time at which proxy expires */
   unsigned long        authDataLength;/* size of authorization data */
   Ptr                  authData;      /* pointer to authorization data */
   unsigned long        proxyLength;   /* max/actual proxy size */
   Ptr                  proxy;         /* buffer where proxy is returned */
   RecordIDPtr          intermediary;  /* record ID of intermediary */
};
typedef struct AuthMakeProxyPB AuthMakeProxyPB;

struct AuthTradeProxyForCredentialsPB {
   AuthDirParamHeader
   AuthIdentity         userIdentity;     /* identity of intermediary */
   RecordIDPtr          recipient;        /* AOCE name of recipient */
   AuthKeyPtr           sessionKey;       /* session key */
   UTCTime              expiry;           /* desired/actual expiration */
   unsigned long        credentialsLength;/* max/actual credentials size */
   Ptr                  credentials;      /* buffer where credentials
                                              are returned */
   unsigned long        proxyLength;      /* actual proxy size */
   Ptr                  proxy;            /* buffer containing proxy */
   RecordIDPtr          principal;        /* record ID of principal */
};
typedef struct AuthTradeProxyForCredentialsPB AuthTradeProxyForCredentialsPB;

struct AuthResolveCreationIDPB {
   AuthDirParamHeader
   RecordIDPtr          userRecord;       /* User record */
   unsigned long        bufferLength;     /* buffer Size */
   Ptr                  buffer;           /* buffer to hold creation IDs */
   unsigned long        totalMatches;     /* total number of matching
                                              names found */
   unsigned long        actualMatches;    /* number of matches returned in
                                              the buffer */
};
typedef struct AuthResolveCreationIDPB AuthResolveCreationIDPB;
```

```
struct AuthGetUTCTimePB {
   AuthDirParamHeader
   PackedRLIPtr      pRLI;          /* packed RLI of the dNode */
   UTCTime           theUTCTime;    /* current UTC(GMT) time in seconds
                                        since 1/1/1904 */
   UTCOffset         theUTCOffset;  /* offset from UTC(GMT) seconds
                                        east of Greenwich */
};
typedef struct AuthGetUTCTimePB AuthGetUTCTimePB;

struct AuthMakeChallengePB {
   AuthDirParamHeader
   AuthKeyPtr        key;                    /* unencrypted session key */
   Ptr               challenge;              /* encrypted challenge */
   unsigned long     challengeBufferLength;  /* length of challenge buffer */
   unsigned long     challengeLength;        /* length of encrypted
                                                 challenge */
};
typedef struct AuthMakeChallengePB AuthMakeChallengePB;

struct AuthMakeReplyPB {
   AuthDirParamHeader
   AuthKeyPtr        key;                    /* unencrypted session key */
   Ptr               challenge;              /* encrypted challenge */
   Ptr               reply;                  /* encrypted reply */
   unsigned long     replyBufferLength;      /* length of challenge buffer */
   unsigned long     challengeLength;        /* length of encrypted
                                                 challenge */
   unsigned long     replyLength;            /* length of encrypted reply */
};
typedef struct AuthMakeReplyPB AuthMakeReplyPB;

struct AuthVerifyReplyPB {
   AuthDirParamHeader
   AuthKeyPtr        key;                    /* unencrypted session key */
   Ptr               challenge;              /* encrypted challenge */
   Ptr               reply;                  /* encrypted reply */
   unsigned long     challengeLength;        /* length of encrypted
                                                 challenge */
   unsigned long     replyLength;            /* length of encrypted reply */
};
typedef struct AuthVerifyReplyPB AuthVerifyReplyPB;
```

```
struct AuthDecryptCredentialsPB {
   AuthDirParamHeader
   AuthIdentity      userIdentity;     /* user's identity */
   RecordIDPtr       initiatorRecord;  /* AOCE name of the initiator */
   AuthKeyPtr        sessionKey;       /* session key */
   UTCTime           expiry;           /* credentials expiration time */
   unsigned long     credentialsLength;/* actual credentials size */
   Ptr               credentials;      /* credentials to be decrypted */
   UTCTime           issueTime;        /* credentials expiration time */
   Boolean           hasIntermediary;  /* if true, an intermediary record
                                           was found in credentials */
   RecordIDPtr       intermediary;     /* record ID of the intermediary */
};
typedef struct AuthDecryptCredentialsPB AuthDecryptCredentialsPB;

struct OCESetupGetDirectoryInfoPB {
   AuthDirParamHeader
   DirectoryNamePtr  directoryName;    /* catalog name */
   DirDiscriminator  discriminator;    /* discriminator for the catalog */
   RecordIDPtr       recordID;         /* record ID for the catalog */
   RStringPtr        nativeName;       /* user name in the catalog world */
   RStringPtr        password;         /* password in the catalog world */
};
typedef struct OCESetupGetDirectoryInfoPB OCESetupGetDirectoryInfoPB;

struct OCESetupAddDirectoryInfoPB {
   AuthDirParamHeader
   CreationID        directoryRecordCID;  /* creation ID for the catalog */
   RecordIDPtr       recordID;            /* record ID for the identity */
   RStringPtr        password;            /* password in the catalog world */
};
typedef struct OCESetupAddDirectoryInfoPB OCESetupAddDirectoryInfoPB;

struct OCESetupChangeDirectoryInfoPB {
   AuthDirParamHeader
   CreationID     directoryRecordCID;     /* creation ID for the catalog */
   RecordIDPtr    recordID;               /* record ID for the identity */
   RStringPtr     password;               /* password in the catalog world */
   RStringPtr     newPassword;            /* new password in the catalog */
};
typedef struct OCESetupChangeDirectoryInfoPB OCESetupChangeDirectoryInfoPB;
```

```
struct OCESetupRemoveDirectoryInfoPB {
   AuthDirParamHeader
   CreationID     directoryRecordCID;     /* creation ID for the catalog */
};
typedef struct OCESetupRemoveDirectoryInfoPB OCESetupRemoveDirectoryInfoPB;
```

Parameter Block Union Structure

```
union AuthParamBlock {
   struct {AuthDirParamHeader}header;
   AuthBindSpecificIdentityPB           bindIdentityPB;
   AuthUnbindSpecificIdentityPB         unbindIdentityPB;
   AuthResolveCreationIDPB              resolveCreationIDPB;
   AuthGetSpecificIdentityInfoPB        getIdentityInfoPB;
   AuthAddKeyPB                         addKeyPB;
   AuthChangeKeyPB                      changeKeyPB;
   AuthDeleteKeyPB                      deleteKeyPB;
   AuthPasswordToKeyPB                  passwordToKeyPB;
   AuthGetCredentialsPB                 getCredentialsPB;
   AuthDecryptCredentialsPB             decryptCredentialsPB;
   AuthMakeChallengePB                  makeChallengePB;
   AuthMakeReplyPB                      makeReplyPB;
   AuthVerifyReplyPB                    verifyReplyPB;
   AuthGetUTCTimePB                     getUTCTimePB;
   AuthMakeProxyPB                      makeProxyPB;
   AuthTradeProxyForCredentialsPB       tradeProxyForCredentialsPB;
   AuthGetLocalIdentityPB               getLocalIdentityPB;
   AuthUnlockLocalIdentityPB            unLockLocalIdentityPB;
   AuthLockLocalIdentityPB              lockLocalIdentityPB;
   AuthAddToLocalIdentityQueuePB        localIdentityQInstallPB;
   AuthRemoveFromLocalIdentityQueuePB   localIdentityQRemovePB;
   AuthSetupLocalIdentityPB             setupLocalIdentityPB;
   AuthChangeLocalIdentityPB            changeLocalIdentityPB;
   AuthRemoveLocalIdentityPB            removeLocalIdentityPB;
   OCESetupAddDirectoryInfoPB           setupDirectoryIdentityPB;
   OCESetupChangeDirectoryInfoPB        changeDirectoryIdentityPB;
   OCESetupRemoveDirectoryInfoPB        removeDirectoryIdentityPB;
   OCESetupGetDirectoryInfoPB           getDirectoryIdentityInfoPB;
};

typedef union AuthParamBlock AuthParamBlock;
typedef AuthParamBlock *AuthParamBlockPtr;
```

Authentication Manager Functions

Key Management

```
pascal OSErr AuthPasswordToKey
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthAddKey     (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthChangeKey  (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthDeleteKey  (AuthParamBlockPtr paramBlock,
                             Boolean async);
```

Local Identity Management

```
pascal OSErr AuthGetLocalIdentity
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthAddToLocalIdentityQueue
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthRemoveFromLocalIdentityQueue
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthSetupLocalIdentity
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthChangeLocalIdentity
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthLockLocalIdentity
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthUnlockLocalIdentity
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr AuthRemoveLocalIdentity
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
```

Specific Identity Management

```
pascal OSErr AuthBindSpecificIdentity
                             (AuthParamBlockPtr paramBlock,
                              Boolean async);
pascal OSErr AuthUnbindSpecificIdentity
                             (AuthParamBlockPtr paramBlock,
                              Boolean async);
pascal OSErr AuthGetSpecificIdentityInfo
                             (AuthParamBlockPtr paramBlock,
                              Boolean async);
```

Credentials Management

```
pascal OSErr AuthGetCredentials
                             (AuthParamBlockPtr paramBlock,
                              Boolean async);
pascal OSErr AuthMakeProxy  (AuthParamBlockPtr paramBlock, Boolean async);
pascal OSErr AuthTradeProxyForCredentials
                             (AuthParamBlockPtr paramBlock,
                              Boolean async);
```

Creation ID Resolution Management

```
pascal OSErr AuthResolveCreationID
                             (AuthParamBlockPtr paramBlock,
                              Boolean async);
```

Time Service

```
pascal OSErr AuthGetUTCTime (AuthParamBlockPtr paramBlock, Boolean async);
```

Non-ASDSP Authentication Utilities

```
pascal OSErr AuthMakeChallenge
                             (AuthParamBlockPtr paramBlock,
                              Boolean async);
pascal OSErr AuthMakeReply  (AuthParamBlockPtr paramBlock, Boolean async);
pascal OSErr AuthVerifyReply
                             (AuthParamBlockPtr paramBlock, Boolean async);
pascal OSErr AuthDecryptCredentials
                             (AuthParamBlockPtr paramBlock,
                              Boolean async);
```

AOCE Setup Catalog Management

```
pascal OSErr OCESetupGetDirectoryInfo
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr OCESetupAddDirectoryInfo
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr OCESetupChangeDirectoryInfo
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
pascal OSErr OCESetupRemoveDirectoryInfo
                            (AuthParamBlockPtr paramBlock,
                             Boolean async);
```

Application-Defined Functions

```
void MyCompletion           (Ptr paramBlk);
pascal Boolean MyNotificationProc
                            (long clientData,
                             AuthLocalIdentityOp callValue,
                             AuthLocalIdentityLockAction actionValue,
                             LocalIdentity identity);
```

Pascal Summary

Constants

```
CONST {values for key sizes}
      kRC4KeySizeInBytes               = 8;        {size of an RC4 key}
      kRefNumUnknown                   = 0;        {dsRefNum specifier}

      {values of AuthLocalIdentityOp}
      kAuthLockLocalIdentityOp         = 1;
      kAuthUnlockLocalIdentityOp       = 2;
      kAuthLocalIdentityNameChangeOp   = 3;

      {values of AuthLocalIdentityLockAction}
      kAuthLockPending                 = 1;
      kAuthLockWillBeDone              = 2;
```

```
    {values of AuthNotifications}
    kNotifyLockBit               = 0;
    kNotifyUnlockBit             = 1;
    kNotifyNameChangeBit         = 2;
    kNotifyLockMask              = $00000001;  {1<<kNotifyLockBit}
    kNotifyUnlockMask            = $00000002;  {1<<kNotifyUnlockBit}
    kNotifyNameChangeMask        = $00000004;  {1<<kNotifyNameChangeBit}
```

Data Types

```
   AuthIdentity   = LongInt;              {unique identifier for an identity}
   LocalIdentity  = AuthIdentity;         {umbrella local identity}

   AuthLocalIdentityOp           = LongInt;
   AuthLocalIdentityLockAction   = LongInt;
   AuthNotifications             = LongInt;
```

Key Structures

```
TYPE
   DESKey =
   RECORD               { a DES key is 8 bytes of data }
      a: LongInt;
      b: LongInt;
   END;

   RC4Key      =  PACKED ARRAY[1..kRC4KeySizeInBytes] OF Byte;
   AuthKeyType =  LongInt;

   AuthKey =
   RECORD               { key type followed by its data }
      keyType: AuthKeyType;
      CASE INTEGER OF
         1: (des: DESKey);
         2: (rc4: RC4Key);
   END;

   AuthKeyPtr     = ^AuthKey;
```

Parameter Block Header

```
   AuthDirParamHeader = RECORD
      qLink:         Ptr;
      reserved1:     LongInt;
      reserved2:     LongInt;
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LongInt;
      reqCode:       Integer;
      reserved:      ARRAY[1..2] OF LongInt;
      serverHint:    AddrBlock;
      dsRefNum:      Integer;
      callID:        LongInt;
      identity:      AuthIdentity;
      gReserved1:    LongInt;
      gReserved2:    LongInt;
      gReserved3:    LongInt;
      clientData:    LongInt;
   END;
```

Parameter Blocks

```
   AuthPasswordToKeyPB = RECORD
      qLink:         Ptr;
      reserved1:     LongInt;
      reserved2:     LongInt;
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LongInt;
      reqCode:       Integer;
      reserved:      ARRAY[1..2] OF LongInt;
      serverHint:    AddrBlock;
      dsRefNum:      Integer;
      callID:        LongInt;
      identity:      AuthIdentity;
      gReserved1:    LongInt;
      gReserved2:    LongInt;
      gReserved3:    LongInt;
      clientData:    LongInt;
      userRecord:    RecordIDPtr;      {User record}
      key:           AuthKeyPtr;
      password:      RStringPtr;       {pointer to the new password string}
   END;

   AuthAddKeyPB = RECORD
      qLink:         Ptr;
      reserved1:     LongInt;
      reserved2:     LongInt;
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LongInt;
      reqCode:       Integer;
      reserved:      ARRAY[1..2] OF LongInt;
      serverHint:    AddrBlock;
      dsRefNum:      Integer;
      callID:        LongInt;
      identity:      AuthIdentity;
      gReserved1:    LongInt;
      gReserved2:    LongInt;
      gReserved3:    LongInt;
      clientData:    LongInt;
      userRecord:    RecordIDPtr;      {User record}
      userKey:       AuthKeyPtr;       {AOCE key for the user}
      password:      RStringPtr;       {pointer to password string}
   END;

   AuthChangeKeyPB = RECORD
      qLink:         Ptr;
      reserved1:     LongInt;
      reserved2:     LongInt;
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LongInt;
      reqCode:       Integer;
      reserved:      ARRAY[1..2] OF LongInt;
      serverHint:    AddrBlock;
      dsRefNum:      Integer;
      callID:        LongInt;
      identity:      AuthIdentity;
      gReserved1:    LongInt;
      gReserved2:    LongInt;
      gReserved3:    LongInt;
      clientData:    LongInt;
      userRecord:    RecordIDPtr;      {User record}
      userKey:       AuthKeyPtr;       {new AOCE key for the user}
      password:      RStringPtr;       {pointer to the new password string}
   END;

   AuthDeleteKeyPB = RECORD
      qLink:         Ptr;
      reserved1:     LongInt;
      reserved2:     LongInt;
```

```
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LongInt;
      reqCode:       Integer;
      reserved:      ARRAY[1..2] OF LongInt;
      serverHint:    AddrBlock;
      dsRefNum:      Integer;
      callID:        LongInt;
      identity:      AuthIdentity;
      gReserved1:    LongInt;
      gReserved2:    LongInt;
      gReserved3:    LongInt;
      clientData:    LongInt;
      userRecord:    RecordIDPtr;            {User record}
   END;

   AuthGetLocalIdentityPB = RECORD
      qLink:            Ptr;
      reserved1:        LongInt;
      reserved2:        LongInt;
      ioCompletion:     ProcPtr;
      ioResult:         OSErr;
      saveA5:           LongInt;
      reqCode:          Integer;
      reserved:         ARRAY[1..2] OF LongInt;
      serverHint:       AddrBlock;
      dsRefNum:         Integer;
      callID:           LongInt;
      identity:         AuthIdentity;
      gReserved1:       LongInt;
      gReserved2:       LongInt;
      gReserved3:       LongInt;
      clientData:       LongInt;
      theLocalIdentity: LocalIdentity;       {local identity}
   END;

   AuthAddToLocalIdentityQueuePB = RECORD
      qLink:         Ptr;
      reserved1:     LongInt;
      reserved2:     LongInt;
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LongInt;
      reqCode:       Integer;
      reserved:      ARRAY[1..2] OF LongInt;
      serverHint:    AddrBlock;
      dsRefNum:      Integer;
      callID:        LongInt;
      identity:      AuthIdentity;
      gReserved1:    LongInt;
      gReserved2:    LongInt;
      gReserved3:    LongInt;
      clientData:    LongInt;
      notifyProc:    NotificationProc;    {notification procedure}
      notifyFlags:   AuthNotifications;   {notification flags}
      appName:       StringPtr;           {name of application to be
                                           returned in Delete/Stop}
   END;

   AuthRemoveFromLocalIdentityQueuePB = RECORD
      qLink:         Ptr;
      reserved1:     LongInt;
      reserved2:     LongInt;
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LongInt;
      reqCode:       Integer;
      reserved:      ARRAY[1..2] OF LongInt;
      serverHint:    AddrBlock;
      dsRefNum:      Integer;
      callID:        LongInt;
      identity:      AuthIdentity;
      gReserved1:    LongInt;
      gReserved2:    LongInt;
      gReserved3:    LongInt;
      clientData:    LongInt;
      notifyProc:    NotificationProc;             {notification procedure}
   END;

   AuthSetupLocalIdentityPB = RECORD
      qLink:         Ptr;
      reserved1:     LongInt;
      reserved2:     LongInt;
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LongInt;
      reqCode:       Integer;
      reserved:      ARRAY[1..2] OF LongInt;
```

```
      serverHint:    AddrBlock;
      dsRefNum:      Integer;
      callID:        LongInt;
      identity:      AuthIdentity;
      gReserved1:    LongInt;
      gReserved2:    LongInt;
      gReserved3:    LongInt;
      clientData:    LongInt;
      aReserved:     LongInt;
      userName:      RStringPtr;              {user name}
      password:      RStringPtr;              {user password}
   END;

   AuthChangeLocalIdentityPB = RECORD
      qLink:         Ptr;
      reserved1:     LongInt;
      reserved2:     LongInt;
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LongInt;
      reqCode:       Integer;
      reserved:      ARRAY[1..2] OF LongInt;
      serverHint:    AddrBlock;
      dsRefNum:      Integer;
      callID:        LongInt;
      identity:      AuthIdentity;
      gReserved1:    LongInt;
      gReserved2:    LongInt;
      gReserved3:    LongInt;
      clientData:    LongInt;
      aReserved:     LongInt;
      userName:      RStringPtr;              {user name}
      password:      RStringPtr;              {current password}
      newPassword:   RStringPtr;              {new password}
   END;

   AuthLockLocalIdentityPB = RECORD
      qLink:            Ptr;
      reserved1:        LongInt;
      reserved2:        LongInt;
      ioCompletion:     ProcPtr;
      ioResult:         OSErr;
      saveA5:           LongInt;
      reqCode:          Integer;
      reserved:         ARRAY[1..2] OF LongInt;
      serverHint:       AddrBlock;
      dsRefNum:         Integer;
      callID:           LongInt;
      identity:         AuthIdentity;
      gReserved1:       LongInt;
      gReserved2:       LongInt;
      gReserved3:       LongInt;
      clientData:       LongInt;
      theLocalIdentity: LocalIdentity; {local identity}
      name:             StringPtr;     {name of the app that denied delete}
   END;

   AuthUnlockLocalIdentityPB = RECORD
      qLink:            Ptr;
      reserved1:        LongInt;
      reserved2:        LongInt;
      ioCompletion:     ProcPtr;
      ioResult:         OSErr;
      saveA5:           LongInt;
      reqCode:          Integer;
      reserved:         ARRAY[1..2] OF LongInt;
      serverHint:       AddrBlock;
      dsRefNum:         Integer;
      callID:           LongInt;
      identity:         AuthIdentity;
      gReserved1:       LongInt;
      gReserved2:       LongInt;
      gReserved3:       LongInt;
      clientData:       LongInt;
      theLocalIdentity: LocalIdentity;            {local identity}
      userName:         RStringPtr;               {user name}
      password:         RStringPtr;               {user password}
   END;

   AuthRemoveLocalIdentityPB = RECORD
      qLink:         Ptr;
      reserved1:     LongInt;
      reserved2:     LongInt;
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LongInt;
      reqCode:       Integer;
      reserved:      ARRAY[1..2] OF LongInt;
```

```
      serverHint:    AddrBlock;
      dsRefNum:      Integer;
      callID:        LongInt;
      identity:      AuthIdentity;
      gReserved1:    LongInt;
      gReserved2:    Longint;
      gReserved3:    LongInt;
      clientData:    LongInt;
      aReserved:     LongInt;
      userName:      RStringPtr;                  {user name}
      password:      RStringPtr;                  {current password}
   END;

   AuthBindSpecificIdentityPB = RECORD
      qLink:         Ptr;
      reserved1:     LongInt;
      reserved2:     LongInt;
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LongInt;
      reqCode:       Integer;
      reserved:      ARRAY[1..2] OF LongInt;
      serverHint:    AddrBlock;
      dsRefNum:      Integer;
      callID:        LongInt;
      identity:      AuthIdentity;
      gReserved1:    LongInt;
      gReserved2:    LongInt;
      gReserved3:    LongInt;
      clientData:    LongInt;
      userIdentity:  AuthIdentity;          {binding identity}
      userRecord:    RecordIDPtr;           {User record}
      userKey:       AuthKeyPtr;            {AOCE key for the user}
   END;

   AuthUnbindSpecificIdentityPB = RECORD
      qLink:         Ptr;
      reserved1:     LongInt;
      reserved2:     LongInt;
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LongInt;
      reqCode:       Integer;
      reserved:      ARRAY[1..2] OF LongInt;
      serverHint:    AddrBlock;
      dsRefNum:      Integer;
      callID:        LongInt;
      identity:      AuthIdentity;
      gReserved1:    LongInt;
      gReserved2:    LongInt;
      gReserved3:    LongInt;
      clientData:    LongInt;
      userIdentity:  AuthIdentity;                 {identity to be deleted}
   END;

   AuthGetSpecificIdentityInfoPB = RECORD
      qLink:         Ptr;
      reserved1:     LongInt;
      reserved2:     LongInt;
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LongInt;
      reqCode:       Integer;
      reserved:      ARRAY[1..2] OF LongInt;
      serverHint:    AddrBlock;
      dsRefNum:      Integer;
      callID:        LongInt;
      identity:      AuthIdentity;
      gReserved1:    LongInt;
      gReserved2:    LongInt;
      gReserved3:    LongInt;
      clientData:    LongInt;
      userIdentity:  AuthIdentity;           {identity of initiator}
      userRecord:    RecordIDPtr;            {User record}
   END;

   AuthGetCredentialsPB = RECORD
      qLink:               Ptr;
      reserved1:           LongInt;
      reserved2:           LongInt;
      ioCompletion:        ProcPtr;
      ioResult:            OSErr;
      saveA5:              LongInt;
      reqCode:             Integer;
      reserved:            ARRAY[1..2] OF LongInt;
      serverHint:          AddrBlock;
      dsRefNum:            Integer;
      callID:              LongInt;
      identity:            AuthIdentity;
      gReserved1:          LongInt;
      gReserved2:          LongInt;
      gReserved3:          LongInt;
      clientData:          LongInt;
      userIdentity:        AuthIdentity;       {identity of initiator}
      recipient:           RecordIDPtr;        {AOCE name of recipient}
      sessionKey:          AuthKeyPtr;         {session key}
      expiry:              UTCTime;            {desired/actual expiration}
      credentialsLength:   LongInt;            {max/actual credentials size}
      credentials:         Ptr;                {credentials buffer}
   END;

   AuthMakeProxyPB = RECORD
      qLink:            Ptr;
      reserved1:        LongInt;
      reserved2:        LongInt;
      ioCompletion:     ProcPtr;
      ioResult:         OSErr;
      saveA5:           LongInt;
      reqCode:          Integer;
      reserved:         ARRAY[1..2] OF LongInt;
      serverHint:       AddrBlock;
      dsRefNum:         Integer;
      callID:           LongInt;
      identity:         AuthIdentity;
      gReserved1:       LongInt;
      gReserved2:       LongInt;
      gReserved3:       LongInt;
      clientData:       LongInt;
      userIdentity:     AuthIdentity;  {identity of principal}
      recipient:        RecordIDPtr;   {AOCE name of recipient}
      firstValid:       UTCTime;       {time at which proxy becomes valid}
      expiry:           UTCTime;       {time at which proxy expires}
      authDataLength:   LongInt;       {size of authorization data}
      authData:         Ptr;           {pointer to authorization data}
      proxyLength:      LongInt;       {max/actual proxy size}
      proxy:            Ptr;           {proxy buffer}
      intermediary:     RecordIDPtr;   {record ID of intermediary}
   END;

   AuthTradeProxyForCredentialsPB = RECORD
      qLink:               Ptr;
      reserved1:           LongInt;
      reserved2:           LongInt;
      ioCompletion:        ProcPtr;
      ioResult:            OSErr;
      saveA5:              LongInt;
      reqCode:             Integer;
      reserved:            ARRAY[1..2] OF LongInt;
      serverHint:          AddrBlock;
      dsRefNum:            Integer;
      callID:              LongInt;
      identity:            AuthIdentity;
      gReserved1:          LongInt;
      gReserved2:          LongInt;
      gReserved3:          LongInt;
      clientData:          LongInt;
      userIdentity:        AuthIdentity;        {identity of intermediary}
      recipient:           RecordIDPtr;         {AOCE name of recipient}
      sessionKey:          AuthKeyPtr;          {session key}
      expiry:              UTCTime;             {desired/actual expiration}
      credentialsLength:   LongInt;             {max/actual credentials size}
      credentials:         Ptr;                 {credentials buffer}
      proxyLength:         LongInt;             {actual proxy size}
      proxy:               Ptr;                 {buffer containing proxy}
      principal:           RecordIDPtr;         {record ID of principal}
   END;

   AuthResolveCreationIDPB = RECORD
      qLink:          Ptr;
      reserved1:      LongInt;
      reserved2:      LongInt;
      ioCompletion:   ProcPtr;
      ioResult:       OSErr;
      saveA5:         LongInt;
      reqCode:        Integer;
      reserved:       ARRAY[1..2] OF LongInt;
      serverHint:     AddrBlock;
      dsRefNum:       Integer;
      callID:         LongInt;
      identity:       AuthIdentity;
      gReserved1:     LongInt;
      gReserved2:     LongInt;
      gReserved3:     LongInt;
      clientData:     LongInt;
      userRecord:     RecordIDPtr;{User record}
      bufferLength:   LongInt;    {buffer size}
```

```
      buffer:        Ptr;        {buffer to hold creation IDs}
      totalMatches:  LongInt;    {total number of matching names found}
      actualMatches: LongInt;    {number of matches returned in the buffer}
   END;

AuthGetUTCTimePB = RECORD
      qLink:         Ptr;
      reserved1:     LongInt;
      reserved2:     LongInt;
      ioCompletion:  ProcPtr;
      ioResult:      OSErr;
      saveA5:        LongInt;
      reqCode:       Integer;
      reserved:      ARRAY[1..2] OF LongInt;
      serverHint:    AddrBlock;
      dsRefNum:      Integer;
      callID:        LongInt;
      identity:      AuthIdentity;
      gReserved1:    LongInt;
      gReserved2:    LongInt;
      gReserved3:    LongInt;
      clientData:    LongInt;
      pRLI:          PackedRLIPtr;  {packed RLI of the dNode}
      theUTCTime:    UTCTime;       {current UTC(GMT) time in seconds
                                     since 1/1/1904}
      theUTCOffset:  UTCOffset;     {offset from UTC(GMT) seconds east
                                     of Greenwich}
   END;

   AuthMakeChallengePB = RECORD
      qLink:                  Ptr;
      reserved1:              LongInt;
      reserved2:              LongInt;
      ioCompletion:           ProcPtr;
      ioResult:               OSErr;
      saveA5:                 LongInt;
      reqCode:                Integer;
      reserved:               ARRAY[1..2] OF LongInt;
      serverHint:             AddrBlock;
      dsRefNum:               Integer;
      callID:                 LongInt;
      identity:               AuthIdentity;
      gReserved1:             LongInt;
      gReserved2:             LongInt;
      gReserved3:             LongInt;
      clientData:             LongInt;
      key:                    AuthKeyPtr;    {unencrypted session key}
      challenge:              Ptr;           {encrypted challenge}
      challengeBufferLength:  LongInt;       {length of challenge buffer}
      challengeLength:        LongInt;       {length of encrypted challenge}
   END;

   AuthMakeReplyPB = RECORD
      qLink:               Ptr;
      reserved1:           LongInt;
      reserved2:           LongInt;
      ioCompletion:        ProcPtr;
      ioResult:            OSErr;
      saveA5:              LongInt;
      reqCode:             Integer;
      reserved:            ARRAY[1..2] OF LongInt;
      serverHint:          AddrBlock;
      dsRefNum:            Integer;
      callID:              LongInt;
      identity:            AuthIdentity;
      gReserved1:          LongInt;
      gReserved2:          LongInt;
      gReserved3:          LongInt;
      clientData:          LongInt;
      key:                 AuthKeyPtr;       {unencrypted session key}
      challenge:           Ptr;              {encrypted challenge}
      reply:               Ptr;              {encrypted reply}
      replyBufferLength:   LongInt;          {length of challenge buffer}
      challengeLength:     LongInt;          {length of encrypted challenge}
      replyLength:         LongInt;          {length of encrypted reply}
   END;

   AuthVerifyReplyPB = RECORD
      qLink:             Ptr;
      reserved1:         LongInt;
      reserved2:         LongInt;
      ioCompletion:      ProcPtr;
      ioResult:          OSErr;
      saveA5:            LongInt;
      reqCode:           Integer;
      reserved:          ARRAY[1..2] OF LongInt;
      serverHint:        AddrBlock;
      dsRefNum:          Integer;
      callID:           LongInt;
      identity:         AuthIdentity;
      gReserved1:       LongInt;
      gReserved2:       LongInt;
      gReserved3:       LongInt;
      clientData:       LongInt;
      key:              AuthKeyPtr;       {unencrypted session key}
      challenge:        Ptr;              {encrypted challenge}
      reply:            Ptr;              {encrypted reply}
      challengeLength:  LongInt;          {length of encrypted challenge}
      replyLength:      LongInt;          {length of encrypted reply}
   END;

   AuthDecryptCredentialsPB = RECORD
      qLink:               Ptr;
      reserved1:           LongInt;
      reserved2:           LongInt;
      ioCompletion:        ProcPtr;
      ioResult:            OSErr;
      saveA5:              LongInt;
      reqCode:             Integer;
      reserved:            ARRAY[1..2] OF LongInt;
      serverHint:          AddrBlock;
      dsRefNum:            Integer;
      callID:              LongInt;
      identity:            AuthIdentity;
      gReserved1:          LongInt;
      gReserved2:          LongInt;
      gReserved3:          LongInt;
      clientData:          LongInt;
      userIdentity:        AuthIdentity;  {user's identity}
      initiatorRecord:     RecordIDPtr;   {AOCE name of the initiator}
      sessionKey:          AuthKeyPtr;    {session key}
      expiry:              UTCTime;       {credentials expiration time}
      credentialsLength:   LongInt;       {actual credentials size}
      credentials:         Ptr;           {credentials to be decrypted}
      issueTime:           UTCTime;       {credentials expiration time}
      hasIntermediary:     Boolean;       {if true, an intermediary record
                                           was found in credentials}
      intermediary:        RecordIDPtr;   {record ID of the intermediary}
   END;
```

```
OCESetupGetDirectoryInfoPB = RECORD
   qLink:         Ptr;
   reserved1:     LongInt;
   reserved2:     LongInt;
   ioCompletion:  ProcPtr;
   ioResult:      OSErr;
   saveA5:        LongInt;
   reqCode:       Integer;
   reserved:      ARRAY[1..2] OF LongInt;
   serverHint:    AddrBlock;
   dsRefNum:      Integer;
   callID:        LongInt;
   identity:      AuthIdentity;
   gReserved1:    LongInt;
   gReserved2:    LongInt;
   gReserved3:    LongInt;
   clientData:    LongInt;
   directoryName: DirectoryNamePtr;    {catalog name}
   discriminator: DirDiscriminator;    {discriminator for the catalog}
   recordID:      RecordIDPtr;         {record ID for the catalog}
   nativeName:    RStringPtr;          {user name in the catalog world}
   password:      RStringPtr;          {password in the catalog world}
END;

OCESetupAddDirectoryInfoPB = RECORD
   qLink:               Ptr;
   reserved1:           LongInt;
   reserved2:           LongInt;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LongInt;
   reqCode:             Integer;
   reserved:            ARRAY[1..2] OF LongInt;
   serverHint:          AddrBlock;
   dsRefNum:            Integer;
   callID:              LongInt;
   identity:            AuthIdentity;
   gReserved1:          LongInt;
   gReserved2:          LongInt;
   gReserved3:          LongInt;
   clientData:          LongInt;
   directoryRecordCID:  CreationID;    {creation ID for the catalog}
   recordID:            RecordIDPtr;   {record ID for the identity}
   password:            RStringPtr;    {password in the catalog world}
END;

OCESetupChangeDirectoryInfoPB = RECORD
   qLink:               Ptr;
   reserved1:           LongInt;
   reserved2:           LongInt;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LongInt;
   reqCode:             Integer;
   reserved:            ARRAY[1..2] OF LongInt;
   serverHint:          AddrBlock;
   dsRefNum:            Integer;
   callID:              LongInt;
   identity:            AuthIdentity;
   gReserved1:          LongInt;
   gReserved2:          LongInt;
   gReserved3:          LongInt;
   clientData:          LongInt;
   directoryRecordCID:  CreationID;    {creation ID for the catalog}
   recordID:            RecordIDPtr;   {record ID for the identity}
   password:            RStringPtr;    {password in the catalog world}
   newPassword:         RStringPtr;    {new password in the catalog}
END;

OCESetupRemoveDirectoryInfoPB = RECORD
   qLink:               Ptr;
   reserved1:           LongInt;
   reserved2:           LongInt;
   ioCompletion:        ProcPtr;
   ioResult:            OSErr;
   saveA5:              LongInt;
   reqCode:             Integer;
   reserved:            ARRAY[1..2] OF LongInt;
   serverHint:          AddrBlock;
   dsRefNum:            Integer;
   callID:              LongInt;
   identity:            AuthIdentity;
   gReserved1:          LongInt;
   gReserved2:          LongInt;
   gReserved3:          LongInt;
   clientData:          LongInt;
   directoryRecordCID:  CreationID;       {creation ID for the catalog}
END;
```

Parameter Block Case Statement

```
AuthParamBlock = RECORD
   CASE INTEGER OF
   1: (header:                      AuthDirParamHeader);
   2: (bindIdentityPB:              AuthBindSpecificIdentityPB);
   3: (unbindIdentityPB:            AuthUnbindSpecificIdentityPB);
   4: (resolveCreationIDPB:         AuthResolveCreationIDPB);
   5: (getIdentityInfoPB:           AuthGetSpecificIdentityInfoPB);
   6: (addKeyPB:                    AuthAddKeyPB);
   7: (changeKeyPB:                 AuthChangeKeyPB);
   8: (deleteKeyPB:                 AuthDeleteKeyPB);
   9: (passwordToKeyPB:             AuthPasswordToKeyPB);
   10:(getCredentialsPB:            AuthGetCredentialsPB);
   11:(decryptCredentialsPB:        AuthDecryptCredentialsPB);
   12:(makeChallengePB:             AuthMakeChallengePB);
   13:(makeReplyPB:                 AuthMakeReplyPB);
   14:(verifyReplyPB:               AuthVerifyReplyPB);
   15:(getUTCTimePB:                AuthGetUTCTimePB);
   16:(makeProxyPB:                 AuthMakeProxyPB);
   17:(tradeProxyForCredentialsPB:  AuthTradeProxyForCredentialsPB);
   18:(getLocalIdentityPB:          AuthGetLocalIdentityPB);
   19:(unLockLocalIdentityPB:       AuthUnlockLocalIdentityPB);
   20:(lockLocalIdentityPB:         AuthLockLocalIdentityPB);
   21:(localIdentityQInstallPB:     AuthAddToLocalIdentityQueuePB);
   22:(localIdentityQRemovePB:      AuthRemoveFromLocalIdentityQueuePB);
   23:(setupLocalIdentityPB:        AuthSetupLocalIdentityPB);
   24:(changeLocalIdentityPB:       AuthChangeLocalIdentityPB);
   25:(removeLocalIdentityPB:       AuthRemoveLocalIdentityPB);
   26:(setupDirectoryIdentityPB:    OCESetupAddDirectoryInfoPB);
   27:(changeDirectoryIdentityPB:   OCESetupChangeDirectoryInfoPB);
   28:(removeDirectoryIdentityPB:   OCESetupRemoveDirectoryInfoPB);
   29:(getDirectoryIdentityInfoPB:  OCESetupGetDirectoryInfoPB);
END;

AuthParamBlockPtr = ^AuthParamBlock;
```

Authentication Manager Functions

Key Management

```
FUNCTION AuthPasswordToKey  (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthAddKey         (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthChangeKey      (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthDeleteKey      (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
```

Local Identity Management

```
FUNCTION AuthGetLocalIdentity
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthAddToLocalIdentityQueue
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthRemoveFromLocalIdentityQueue
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthSetupLocalIdentity
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthChangeLocalIdentity
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthLockLocalIdentity
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthUnlockLocalIdentity
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthRemoveLocalIdentity
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
```

Specific Identity Management

```
FUNCTION AuthBindSpecificIdentity
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthUnbindSpecificIdentity
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthGetSpecificIdentityInfo
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
```

Credentials Management

```
FUNCTION AuthGetCredentials  (paramBlock: AuthParamBlockPtr;
                              async: Boolean): OSErr;
FUNCTION AuthMakeProxy       (paramBlock: AuthParamBlockPtr;
                              async: Boolean): OSErr;
FUNCTION AuthTradeProxyForCredentials
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
```

Creation ID Resolution Management

```
FUNCTION AuthResolveCreationID
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
```

Time Service

```
UNCTION AuthGetUTCTime      (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
```

Non-ASDSP Authentication Utilities

```
FUNCTION AuthMakeChallenge  (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthMakeReply      (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthVerifyReply    (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION AuthDecryptCredentials
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
```

AOCE Setup Catalog Management

```
FUNCTION OCESetupGetDirectoryInfo
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION OCESetupAddDirectoryInfo
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION OCESetupChangeDirectoryInfo
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
FUNCTION OCESetupRemoveDirectoryInfo
                            (paramBlock: AuthParamBlockPtr;
                             async: Boolean): OSErr;
```

Application-Defined Routines

```
PROCEDURE MyCompletion      (paramBlock: AuthParamBlockPtr);
FUNCTION NotificationProc   (clientData: LongInt;
                             callValue: AuthLocalIdentityOp;
                             actionValue: AuthLocalIdentityLockAction;
                             identity: LocalIdentity): Boolean;
```

Assembly-Language Summary

Trap Macros

Trap Macro Requiring Routine Selectors

`_oceTBDispatch`

Selector	Routine
$0200	`AuthBindSpecificIdentity`
$0201	`AuthUnbindSpecificIdentity`
$0202	`AuthResolveCreationID`
$0203	`AuthGetSpecificIdentityInfo`
$0204	`AuthGetLocalIdentity`
$0205	`AuthAddToLocalIdentityQueue`
$0206	`AuthRemoveFromLocalIdentityQueue`
$0207	`AuthAddKey`
$0208	`AuthChangeKey`

Selector	Routine
$0209	`AuthDeleteKey`
$020A	`AuthPasswordToKey`
$020B	`AuthGetCredentials`
$020C	`AuthDecryptCredentials`
$020D	`OCESetupRemoveDirectoryInfo`
$020E	`OCESetupGetDirectoryInfo`
$020F	`AuthMakeChallenge`
$0210	`AuthMakeReply`
$0211	`AuthVerifyReply`
$0212	`AuthMakeProxy`
$0213	`AuthTradeProxyForCredentials`
$0214	`AuthUnlockLocalIdentity`
$0215	`AuthLockLocalIdentity`
$0216	`AuthSetupLocalIdentity`
$0217	`AuthChangeLocalIdentity`
$0218	`AuthRemoveLocalIdentity`
$0219	`OCESetupAddDirectoryInfo`
$021A	`AuthGetUTCTime`
$021B	`OCESetupChangeDirectoryInfo`

Result Codes

Result codes in the range of –1540 to –1609 are reserved for the Authentication Manager. Routines may also return result codes from other AOCE managers and standard Macintosh result codes such as `noErr` 0 (No error) and `fnfErr` –43 (File not found).

`noErr`	0	No error
`kOCEParamErr`	–50	Parameter error
`kOCEReadAccessDenied`	–1540	Read access denied
`kOCEWriteAccessDenied`	–1541	Write access denied
`kOCEAccessRightsInsufficient`	–1542	Stream needs to be authenticated, or not authorized, or someone other than agent trying to TPFC, or problem in server-to-server authentication
`kOCEUnsupportedCredentialsVersion`	–1543	Can’t read this version of the credentials
`kOCECredentialsProblem`	–1544	Couldn’t decrypt credentials

`kOCECredentialsImmature`	–1545	Credentials not yet valid
`kOCECredentialsExpired`	–1546	Current time is later than credentials expiration time
`kOCEProxyImmature`	–1547	Proxy not yet valid
`kOCEProxyExpired`	–1548	Current time is later than proxy expiration time
`kOCEDisallowedRecipient`	–1549	Recipient record ID does not appear in proxy
`kOCENoKeyFound`	–1550	No key was found
`kOCEPrincipalKeyNotFound`	–1551	Couldn't decode proxy because principal has no key
`kOCERecipientKeyNotFound`	–1552	The recipient key was not found
`kOCEAgentKeyNotFound`	–1553	Intermediary's key not found
`kOCEKeyAlreadyRegistered`	–1554	A key already exists
`kOCEMalFormedKey`	–1555	Key not derived properly from password
`kOCEUndesirableKey`	–1556	Password too short or resulting key is undesirable
`kOCEWrongIdentityOrKey`	–1557	Incorrect key for client
`kOCEInitiatorKeyProblem`	–1558	No key, or initiator's key changed
`kOCEBadEncryptionMethod`	–1559	The specified encryption method is not supported
`kOCELocalIdentityDoesNotExist`	–1560	Local identity has not been set up
`kOCELocalAuthenticationFail`	–1561	Local identity locked
`kOCELocalIdentitySetupExists`	–1562	Local identity setup exists, use `AuthChangeLocalIdentity` instead
`kOCEDirectoryIdentitySetupExists`	–1563	Catalog has already been set up
`kOCEDirectoryIdentitySetupDoesNotExist`	–1564	Catalog has not been set up
`kOCENotLocalIdentity`	–1565	You cannot unbind a local identity
`kOCENoMoreIDs`	–1566	Identity table is full
`kOCEUnknownID`	–1567	Identity passed is not valid
`kOCEOperationDenied`	–1568	Local identity operation denied
`kOCEAmbiguousMatches`	–1569	Ambiguous matches found in resolving CIDs
`kOCENoASDSPWorkSpace`	–1570	No ASDSP workspace passed
`kOCEAuthenticationTrouble`	–1571	Reply incorrect for the challenge sent
`kOCENotLocal`	–1610	Internal AOCE error
`kOCETargetDirectoryInaccessible`	–1613	Catalog server not responding
`kOCENoSuchDNode`	–1615	The dNode was not found
`kOCEBadRecordID`	–1617	Name and type incorrect for creation ID
`kOCENoSuchRecord`	–1618	No such record
`kOCEMoreData`	–1623	Buffer was too small to hold all available data
`kOCEStreamCreationErr`	–1625	An error occurred in creating the stream
`kOCEDirectoryNotFoundErr`	–1630	Catalog was not found in the list
`kOCEOCESetupRequired`	–1633	Setup of local identity required
`kOCELengthError`	–1637	The supplied buffer was too small