This chapter describes how you install and configure an Apple Open Collaboration Environment (AOCE) catalog service access module (CSAM) and a personal messaging service access module (personal MSAM). It also describes how any messaging service access module, either personal or server type, obtains address information from a user.

You need to read this chapter if you are developing a catalog service access module or a personal messaging service access module to work with the PowerTalk software. For information on initializing a server MSAM, see the chapter "Messaging Service Access Modules" in this book and the *PowerShare System Manager's Guide*.

This chapter assumes you have already read one or both of the chapters that describe service access modules, "Catalog Service Access Modules" and "Messaging Service Access Modules," both in this book. To use this chapter, you must also know how to write an AOCE template. The chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces* describes how to write an AOCE template. In addition, you must have a general understanding of AOCE catalogs and the Catalog Manager application programming interface (API). See the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* for information about AOCE catalogs.

This chapter begins with a brief introduction to the setup process. It then describes the types of records in the PowerTalk Setup catalog and the setup and address AOCE templates. Then it explains how you can

- set up a combined CSAM/MSAM
- set up a CSAM
- set up an MSAM
- modify existing CSAM or MSAM configuration information
- add, modify, and remove address information

Introduction to SAM Setup

A user of PowerTalk software who wants to add a catalog or messaging service uses the PowerTalk Key Chain to add and configure the service. The PowerTalk Key Chain, in turn, uses special AOCE templates that you provide with your service access module to supply information pages that let the user enter and modify setup information.

Each CSAM or personal MSAM requires a setup template. A ***setup template*** is a set of AOCE templates that allow a user to install and configure a service access module. In addition, both personal and server MSAMs require an ***address template*** that allows a user to enter address information into a User record.

The templates provide the only human interface for CSAMs and personal MSAMs. Server MSAMs, being foreground applications, have their own user interface. However, the address template is the mechanism through which a server MSAM obtains the address information that it requires.

There are three types of CSAM and personal MSAM files:

- CSAM only, or stand-alone CSAM (file type `'dsam'`)
- MSAM only, or stand-alone MSAM (file type `'msam'`)
- combined CSAM/MSAM (file type `'csam'`)

Note
The abbreviation "dsam"—found in the `'dsam'` file type and in function names and data structures in the AOCE interface files—stands for "directory service access module," the name used for catalog service access modules in early versions of the AOCE software. The `'csam'` file type is so named because it implements a "combined service access module." Therefore, a file of type `'dsam'` implements a CSAM, and a file of type`'csam'` implements both a CSAM and an MSAM. ◆

A CSAM-only file contains the CSAM driver and a setup template. It might also contain additional templates to allow items in its catalogs to be viewed in the Finder. An MSAM-only file contains a background application (the MSAM itself), a setup template, and an address template. A combined CSAM/MSAM file contains a CSAM driver, an MSAM background application, a setup template, and an address template (and possibly other templates to allow items in its catalogs to be viewed in the Finder).

A server MSAM application file (file type `'APPL'`) contains an address template among its resources.

The setup templates add information to the PowerTalk Setup catalog. The ***PowerTalk Setup catalog*** is a personal catalog named "PowerTalk Setup Preferences" and kept in the Preferences folder in the System Folder. The Setup catalog stores information about the catalog and messaging services that are available to the principal user of a given Macintosh computer.

The following sections describe how you use routines from the MSAM and Catalog Manager APIs and AOCE utility routines to install and configure a service access module. The reference section describes the records in the PowerTalk Setup catalog, the AOCE templates that create and modify those records, and the resources required for these templates.

About Personal MSAMs and Addresses

When a user sends an AppleMail letter, the AOCE software examines the recipients in the mailer and determines how to route the letter based on the catalog names in the recipient addresses. For example, if an address contains the name of a PowerShare catalog, the AOCE toolbox hands the letter to the PowerShare router. For this reason, every personal MSAM must have an associated catalog name, even if there is no CSAM associated with the MSAM. A CSAM provides the user with the ability to browse an external catalog. The catalog name associated with an MSAM, by contrast, is used by the AOCE software for routing the letter. It can be the name of an external catalog for which the user has a CSAM, but need not be.

Note
In the current version of the PowerTalk system software, each personal MSAM must be associated with a unique catalog name. In future versions of PowerTalk, two or more personal MSAMs may be able to use the same catalog name for routing purposes. In that case, the AOCE software would look at the extension type in the address as well as the catalog name. (Addresses and address extension types are described in the chapter "Messaging Service Access Modules" in this book.) ◆

Your address template must allow the user to enter addressing information and be capable of formulating an address in the format required by AOCE.

For incoming mail, the personal MSAM has no responsibility to forward mail to other recipients. However, to be most useful to the user, your personal MSAM should translate the addresses from its own external messaging system into addresses intelligible to AOCE, including the catalog name and the address extension information, and place them in the mailer. Then the user can use these addresses for replies to the letter and can drag them out of the mailer into a personal catalog or information card for future use.

Adding Catalog and Mail Services

This section describes what your setup template must do to add catalog and mail services to a user's AOCE system. It first discusses what your setup template must do when you provide a combined CSAM/MSAM. Then it explains what the setup template must do when you provide only a CSAM or an MSAM, but not both.

The process of adding a service requires actions from the user, the PowerTalk Key Chain, and your setup template. This section describes the user and Key Chain actions only to the extent that it clarifies the actions your setup template must take. If you want more detail on the user actions, see the *PowerTalk User's Guide*.

As you will see in the following sections, before you can add a service of any type (catalog, mail, or combined), you must have added the SAM itself. The procedure for adding a SAM is covered in the description of adding the relevant service.

Each section identifies the records and attributes in the PowerTalk Setup catalog that your setup template creates or manipulates in the course of adding a given service, and it identifies the functions you use to do that. Some of these functions are described in the other chapters in this book. Others are described in the chapters "AOCE Utilities," "AOCE Templates," "Catalog Manager," and "Authentication Manager" in *Inside Macintosh: AOCE Application Interfaces*.

AOCE record and attribute types that are discussed in this section are often identified by an index constant. See the chapter "AOCE Utilities" for an explanation of indexed record and attribute types.

Adding a Combined Service

This section tells you how to add a combined catalog and mail service. Read this section if you are providing both a CSAM and an MSAM. It explains what your setup template must do to allow a user to add and configure a combined mail and catalog service.

The setup template for a combined MSAM and CSAM includes a main aspect template for a Combined record. Your code resource for the Combined record must create a CSAM record and check whether an MSAM record already exists. If the MSAM record does not exist, the template's code resource must create that record as well.

Listing 4-1 illustrates a setup template for a combined catalog and mail service.

Listing 4-1 Combined catalog and mail service setup template

```
#define SystemSevenOrLater 1

#include "Types.r"
#include "OCETemplates.h"
#include "OCE.r"

#define kSurfWriterAspect        1000
#define kSurfWriterInfoPage      1250

#define kSignature               'WAVE'
#define kServiceRecordType       kCombinedRecTypeBody "WAVE"
#define kAspectName              "SurfWriter_Service_Aspect"
#define kInfoPageName            "SurfWriter_Service_Info_Page"

#define kWindowWidth             259
#define kWindowHeight            200
#define kZeroRect                {0, 0, 0, 0}
#define kDoubleLineLeft          kDETSubpageIconLeft
#define kDoubleLineRight         kWindowWidth - kDETSubpageIconLeft
#define kDoubleLineTop           kTopBorder + kFieldHeight + 8
#define kDoubleLineBottom        kDoubleLineTop + 1
#define kDoubleLineRect          {kDoubleLineTop, kDoubleLineLeft,\
                                 kDoubleLineBottom, kDoubleLineRight}

resource 'deta' (kSurfWriterAspect,purgeable)
{
   0, dropCheckConflicts, isMainAspect
};
```

```
resource 'rstr' (kSurfWriterAspect + kDETTemplateName, purgeable)
{
   kAspectName
};

resource 'rstr' (kSurfWriterAspect + kDETRecordType, purgeable)
{
   kServiceRecordType
};

resource 'rstr' (kSurfWriterAspect + kSAMAspectKind, purgeable)
{
   "Full SurfWriter Service"
};

resource 'rstr' (kSurfWriterAspect + kDETAspectKind, purgeable)
{
   "Combined SurfWriter Service"
};

resource 'rstr' (kSurfWriterAspect + kDETAspectName, purgeable) {
   "unconfigured SurfWriter Service"
};

resource 'rstr' (kSurfWriterAspect + kSAMAspectUserName, purgeable) {
   "SurfWriter User"
};

resource 'sami' (kSurfWriterAspect + kSAMAspectSlotCreationInfo, purgeable)
{
   2,                                    // max number of catalogs/slots
   kSignature,                           // catalog signature, MSAM type
   servesMSAM,                           // an MSAM template
   servesDSAM,                           // a CSAM template
   "SurfWriter Combined Service ",       // display when user clicks Add
   "untitled combined SurfWriter Service" // new record name
};
```

```
// Custom window

resource 'detw' (kSurfWriterAspect + kDETAspectInfoPageCustomWindow,
                 purgeable)
   {
   {-1, -1, kWindowHeight, kWindowWidth},
   includePopup
   };

// Include code resource

include "SurfWriterCode" 'detc'(0) as
    'detc'(kSurfWriterAspect+kDETAspectCode, purgeable);

// Include icons

include "SurfWriterIcons" 'ICN#'(0) as
                  'ICN#'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'icl4'(0) as
                  'icl4'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'icl8'(0) as
                  'icl8'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics#'(0) as
                  'ics#'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics4'(0) as
                  'ics4'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics8'(0) as
                  'ics8'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'SICN'(0) as
                  'SICN'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);

//---------------------------------------------------------------------------
// Info-page

resource 'deti' (kSurfWriterInfoPage, purgeable)
{
   kDefaultSortIndex,
   kZeroRect,
   noSelectFirstText,
   {
      kDETNoProperty, kDETNoProperty, kSurfWriterInfoPage;
   },
   {
```

```
   }
};

resource 'rstr' (kSurfWriterInfoPage + kDETTemplateName, purgeable) {
   kInfoPageName
};

resource 'rstr' (kSurfWriterInfoPage + kDETRecordType, purgeable) {
   kServiceRecordType
};

resource 'rstr' (kSurfWriterInfoPage + kDETInfoPageName, purgeable) {
   "SurfWriter Combined Service"
};

resource 'rstr' (kSurfWriterInfoPage + kDETInfoPageMainViewAspect,
                 purgeable) {
   kAspectName
};

resource 'detv' (kSurfWriterInfoPage, "subpageview", purgeable)
   {
   {
   kDoubleLineRect, kDETNoFlags, kDETNoProperty,
   Box { kDETBoxIsGrayed };

   kDETSubpageIconRect, kDETNoFlags, kDETAspectMainBitmap,
   Bitmap { kDETLargeIcon };
   };
   };
```

During system initialization, the Key Chain loads all setup templates. As specified by the `'sami'` resource in Listing 4-1, the Key Chain offers the user the choice “SurfWriter Combined Service” when the user clicks the Add button. When the user selects this choice, the Key Chain adds a new record to the PowerTalk Setup catalog. As specified by Listing 4-1, this record’s name is “untitled combined SurfWriter Service,” and its type is “aoce CombinedWAVE”. The Key Chain also writes four attributes into this record, as follows:

- An associated catalog attribute that points to the record that contains information about the catalog associated with this service. In the case of a combined service, this attribute points back to the Combined record itself.
- An associated mail service attribute that points to the record that contains information about the mail slot associated with this service. In the case of a combined service, this attribute points back to the Combined record itself.

- An attribute of type “aoce Unconfigured” (attribute type index `kUnconfiguredAttrTypeNum`) indicating that the service has not yet been set up.
- A version attribute that contains the version number of the Key Chain at the time it created the record.

The Key Chain adds a line to the Key Chain window representing the new record. The line includes the key icon used by the Key Chain, the service name you specified in the `kDETAspectName` resource (“unconfigured SurfWriter Service” in Listing 4-1), and whatever default values your template provides for the Name and Kind fields in the `kSAMAspectUserName` and `kSAMAspectKind` resources (“SurfWriter User” and “Full SurfWriter Service” in Listing 4-1).

Your code resource should create no new records or attributes at this time. However, when the user opens the Key Chain entry, the Key Chain opens your information page and calls your code resource with the `kDETcmdInstanceInit` routine selector. Your setup template must follow the steps in the following two sections in order to set up a combined mail and catalog service.

Adding the Catalog Service

To add a catalog service to a combined service, you must write certain attributes and also activate the catalog. The attributes in the Combined record are summarized in Table 4-6 on page 4-70. Use the Catalog Manager function `DirAddAttributeValue` to add attributes to a record. The function is described in the chapter “Catalog Manager” in *Inside Macintosh: AOCE Application Interfaces*; see the `DoAddAttribute` function on page 4-18 in Listing 4-2 for an example of its use.

Note
If you are adding a catalog service only, you follow these same steps but add and modify attributes in the Catalog record rather than in the Combined record. See “Adding a Catalog Service Only” beginning on page 4-28 for more information. ◆

1. Write the comment attribute. This is a string that you can use for any purpose. An application or template code resource can read this string by calling the `DirGetExtendedDirectoriesInfo` function; see the chapter “Catalog Manager” in *Inside Macintosh: AOCE Application Interfaces* for a description of this function.
2. Write the real name attribute, containing the external name of the catalog. This name is for your own use; it is not read by the AOCE software. An application or template code resource can read this name by calling the `DirGetExtendedDirectoriesInfo` function. Whereas the catalog name you provide to the `DirAddDSAMDirectory` function (see step 5) must be unique within the AOCE system, the “real” name need not be. For example, the user might have accounts on two different SurfWriter mail servers. Each would have to have a distinct name for display in the Key Chain, but the Combined records for both could contain the name “SurfWriter Mail” for the real name attribute.

3. If you wish, you can write a private data attribute. This attribute can contain binary data of any length (up to the maximum length of an attribute) and is for your own use. For example, you can store information about address formats for use by your address template. Your application or template code resource can read this data by calling the `DirGetExtendedDirectoriesInfo` function.
4. Call the `DirAddDSAM` function, passing it the CSAM's name and signature and the file system specification of the CSAM file. You can use the `kDETcmdGetTemplateFSSpec` template callback function to obtain the file system specification. (Template callback functions are described in "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces*.) The `DirAddDSAM` function creates a CSAM record, starts the CSAM driver (if it's not already running), and returns the creation ID of the CSAM record. Your CSAM driver's Open routine should call the `DirInstantiateDSAM` function at this time, providing it the entry point into your CSAM.
5. Determine the name and discriminator of the catalog and then call the `DirAddDSAMDirectory` function, passing it the CSAM record's creation ID, the Combined record's creation ID, the catalog's name and discriminator, and the capability flags that indicate the abilities of the catalog and CSAM. The catalog name must be unique; the record name is the same as the catalog name and cannot be changed. You can use the `kDETcmdGetDSSpec` template callback function to determine the creation ID of the Combined record. The `DirAddDSAMDirectory` function writes to your Combined record the capability flags, catalog discriminator, and parent CSAM attributes. It then calls your CSAM's catalog service routine with the `kDirAddDSAMDirectory` request code so that the CSAM receives all the information about the catalog that you passed to the `DirAddDSAMDirectory` function. The CSAM can use the record's creation ID to read any of the attributes in the Combined record.
6. If the catalog requires a user name and password, call the `OCESetupAddDirectoryInfo` function to add that information to the Combined record. You provide the Combined record's creation ID, the record ID that represents the user, and the password. The function encrypts the password. You can change this information later by calling the `OCESetupChangeDirectoryInfo` function and extract it by calling the `OCESetupGetDirectoryInfo` function. These functions are described in the chapter "Authentication Manager" in *Inside Macintosh: AOCE Application Interfaces*.
7. If the name of the User record does not correspond to the user's account name in the external catalog, you can also call the `DirAddAttributeValue` function to add a native name attribute. This attribute contains the user's name or account name in the external catalog. This name is for your own use; it is not read by the AOCE software. An application or template code resource can read this name by calling the `OCESetupGetDirectoryInfo` function.

Adding the Mail Service

Once you've added the catalog service, you can add the mail service. Perform the following steps to do so.

8. Determine if an MSAM record exists for your MSAM. You can do this by comparing your MSAM file's file ID with the file ID stored in the gateway file ID attribute in each MSAM record currently in the PowerTalk Setup catalog.
9. If no MSAM record exists for your MSAM, create one and add a version attribute and a gateway file ID attribute to it.

   Listing 4-2 shows a function that compares file IDs, creates the MSAM record if necessary (adding the two attributes), and returns the record's creation ID.

Listing 4-2 Matching an MSAM file ID

```
#define kEnumBufferSize 1024
#define kInitialLookupBuffer 256

struct LookupInfo {
   unsigned long  fileID;
   Boolean     found;
};

#ifndef __cplusplus
typedef struct LookupInfo LookupInfo;
#endif

struct EnumInfo {
   short          setupRef;
   unsigned long  fileID;
   CreationID     msamCID;
   DirEnumSpec    lastEnumSpec;
   RString        name;
   RString        type;
};

#ifndef __cplusplus
typedef struct EnumInfo EnumInfo;
#endif

/* Return the given file's file ID. */

OSErr DoGetIDFromFSSpec(const FSSpec *spec, unsigned long *id)
   {
   OSErr err;
```

```
   CInfoPBRec pb;

   pb.dirInfo.ioCompletion = nil;
   pb.dirInfo.ioVRefNum = spec->vRefNum;
   pb.dirInfo.ioNamePtr = spec->name;
   pb.dirInfo.ioFDirIndex = 0;
   pb.dirInfo.ioDrDirID = spec->parID;
   err = PBGetCatInfoSync(&pb);
   if (err == noErr)
      {
      *id = pb.dirInfo.ioDrDirID;
      }
   return err;
   }

/* Return the dsRefNum of the Setup catalog. */

OSErr DoGetSetupDirectoryRefNum(short *refNum)
   {
   OSErr err;
   DirParamBlock dspb;

   err = DirGetOCESetupRefNum(&dspb, false);

   if (err == noErr)
      {
      *refNum = dspb.header.dsRefNum;
      }

   return err;
   }


/* Does this record have the matching file ID as an attribute? */

pascal Boolean DoLookupCB(long lInfo, const Attribute *thisAttribute)
   {
   LookupInfo* info = (LookupInfo*) lInfo;

   if (info->fileID == * (long*) thisAttribute->value.bytes)
      {
      info->found = true;
      }
```

```
   /* Stop the parse once you find an attribute value with the matching
      file ID. */
   return info->found;
   }

/* Do a lookup into the given record in the Setup catalog to
    see if its kGatewayFileIDAttrTypeNum attribute stores the given
    file ID. */

Boolean DoRecordHasFileID(const LocalRecordID *lrid, short setupRef,
                          unsigned long fileID)
   {
   OSErr err;
   LookupInfo info;
   DirParamBlock dspb;
   RecordID rid;
   RecordIDPtr recordList[1];
   AttributeTypePtr attrTypeList[1];
   Boolean includeStartingPoint = false;
   unsigned long bufferSize = kInitialLookupBuffer;
   void *dataBuffer;

   info.fileID = fileID;
   info.found = false;

   rid.local = *lrid;   /* The lookup requires a RID, not a local RID. */
   rid.rli = nil;

   recordList[0] = &rid;
   attrTypeList[0] = OCEGetIndAttributeType(kGatewayFileIDAttrTypeNum);


/* Create a buffer that's big enough to get at least one attribute value. */

   dataBuffer = NewPtr(bufferSize);
   err = MemError();
   if (err == noErr)
      {
      *(long *)&dspb.lookupGetPB.serverHint = 0;
      dspb.lookupGetPB.dsRefNum = setupRef;
      dspb.lookupGetPB.aRecordList = recordList;
      dspb.lookupGetPB.attrTypeList = attrTypeList;
      dspb.lookupGetPB.recordIDCount = 1;
```

```
      dspb.lookupGetPB.attrTypeCount = 1;
      dspb.lookupGetPB.includeStartingPoint = false;
      dspb.lookupGetPB.getBuffer = dataBuffer;
      dspb.lookupGetPB.getBufferSize = bufferSize;
      dspb.lookupGetPB.startingRecordIndex = 1;
      dspb.lookupGetPB.startingAttrTypeIndex = 1;
      OCESetCreationIDtoNull(&dspb.lookupGetPB.startingAttribute.cid);

      dspb.lookupParsePB.clientData = &info;
      dspb.lookupParsePB.eachRecordID = nil;
      dspb.lookupParsePB.eachAttrType = nil;
      dspb.lookupParsePB.eachAttrValue = DoLookupCB;

      do
         {
         err = DirLookupGet(&dspb,false);

         if ((err == noErr) || (err == kOCEMoreData))
            {
            err = DirLookupParse(&dspb,false);
            }
         } while (err == kOCEMoreData);

      DisposePtr((Ptr) dataBuffer);
      }

   return info.found;
   }


/* Check whether the current record is your MSAM record. To do so, check
   whether the current record's kGatewayFileIDAttrTypeNum attribute matches
   the file ID of your MSAM file. */

pascal Boolean DoEnumCB(long eData, const DirEnumSpec *enumSpecPtr)
   {
   Boolean found;
   EnumInfo* enumData = (EnumInfo*) eData;

   /* First save your current DirEnumSpec so that you can continue
       if the buffer couldn't hold all the records. */
   BlockMove(enumSpecPtr, &enumData->lastEnumSpec, sizeof(DirEnumSpec));
   OCECopyRString(enumSpecPtr->u.recordIdentifier.recordName,
```

```
    &enumData->name, kRStringMaxBytes);
   OCECopyRString(enumSpecPtr->u.recordIdentifier.recordType,
    &enumData->type, kRStringMaxBytes);
   enumData->lastEnumSpec.u.recordIdentifier.recordName = &enumData->name;
   enumData->lastEnumSpec.u.recordIdentifier.recordType = &enumData->type;

   /* Does this record have the matching file ID? */
   found = DoRecordHasFileID(&enumSpecPtr->u.recordIdentifier,
                             enumData->setupRef, enumData->fileID);

   /* If so, save its creation ID. */
   if (found)
      {
      OCECopyCreationID(&enumSpecPtr->u.recordIdentifier.cid,
       &enumData->msamCID);
      }

   /* Stop the parse once you find a record with the matching file ID. */
   return found;
   }

/* Enumerate all MSAM records in the Setup catalog, looking
    for one that corresponds to the MSAM with the given file ID. */

OSErr DoFindMSAMRecordWithFileID(short setupRef, unsigned long fileID,
                                 CreationID *msamCID)
   {
   OSErr err;
   DirParamBlock dspb;
   void *buffer;
   RString *recordType;
   EnumInfo enumData;

   buffer = NewPtr(kEnumBufferSize);
   err = MemError();

   if (err == noErr)
      {
      recordType = OCEGetIndRecordType(kMSAMRecTypeNum);

      enumData.fileID = fileID;
      enumData.setupRef = setupRef;
      OCESetCreationIDtoNull(&enumData.msamCID);
```

```
      *(long *)&dspb.enumerateGetPB.serverHint = 0;
      dspb.enumerateGetPB.dsRefNum = setupRef;
      dspb.enumerateGetPB.clientData = &enumData;
      dspb.enumerateGetPB.aRLI = nil;
      dspb.enumerateGetPB.startingPoint = nil;
      dspb.enumerateGetPB.sortBy = kSortByName;
      dspb.enumerateGetPB.sortDirection = kSortForwards;
      dspb.enumerateGetPB.nameMatchString = nil;
      dspb.enumerateGetPB.typesList = &recordType;
      dspb.enumerateGetPB.typeCount = 1;
      dspb.enumerateGetPB.enumFlags = kEnumDistinguishedNameMask;
      dspb.enumerateGetPB.includeStartingPoint = false;
      dspb.enumerateGetPB.matchNameHow = kMatchAll;
      dspb.enumerateGetPB.matchTypeHow = kExactMatch;
      dspb.enumerateGetPB.getBuffer = buffer;
      dspb.enumerateGetPB.getBufferSize = kEnumBufferSize;

      dspb.enumerateParsePB.eachEnumSpec = DoEnumCB;

      do
         {
         err = DirEnumerateGet(&dspb, false);
         if ((err == noErr) || (err == kOCEMoreData))
            {
            err = DirEnumerateParse(&dspb, false);
            }
         dspb.enumerateGetPB.startingPoint = &enumData.lastEnumSpec;
         } while (err == kOCEMoreData);

      DisposPtr((Ptr) buffer);
      }

   OCECopyCreationID(&enumData.msamCID, msamCID);

   return err;
   }

/* Create an MSAM record, returning the record's creation ID. */

OSErr DoCreateMSAMRecord(short setupRef, CreationID *msamCID)
   {
   OSErr err;
   RString name;
```

```
   RecordID rid;
   DirParamBlock dspb;

   OCEPToRString("\pMy MSAM", smRoman, &name, kRStringMaxBytes);
   rid.local.recordName = &name;
   rid.local.recordType = OCEGetIndRecordType(kMSAMRecTypeNum);
   OCESetCreationIDtoNull(&rid.local.cid);
   rid.rli = nil;

   *(long *)&dspb.addRecordPB.serverHint = 0;
   dspb.addRecordPB.dsRefNum = setupRef;
   dspb.addRecordPB.aRecord = &rid;
   dspb.addRecordPB.allowDuplicate = true;

   err = DirAddRecord(&dspb, false);

   OCECopyCreationID(&rid.local.cid, msamCID);

   return err;
   }

/* Add an attribute value to the given record in the Setup catalog. */

OSErr DoAddAttribute(short setupRef,
            const CreationID *recordCID,
            const AttributeType *attrType,
            unsigned long length,
            Ptr bytes)
   {
   OSErr err;
   RecordID rid;
   Attribute attr;
   DirParamBlock dspb;

   rid.local.recordName = nil;
   rid.local.recordType = nil;
   OCECopyCreationID(recordCID, &rid.local.cid);
   rid.rli = nil;

   OCECopyRString((RString*) attrType, (RString*) &attr.attributeType,
                  kAttributeTypeMaxBytes);
   OCESetCreationIDtoNull(&attr.cid);
   attr.value.tag = typeBinary;
```

```
   attr.value.dataLength = length;
   attr.value.bytes = bytes;

   *(long *)&dspb.addAttributeValuePB.serverHint = 0;
   dspb.addAttributeValuePB.dsRefNum = setupRef;
   dspb.addAttributeValuePB.aRecord = &rid;
   dspb.addAttributeValuePB.attr = &attr;

   err = DirAddAttributeValue(&dspb, false);

   return err;
   }

/* Given an FSSpec representing an MSAM file, return the corresponding
    MSAM record's creation ID, creating the MSAM record if necessary. */

OSErr DoGetMSAMCreationID(const FSSpec *spec, CreationID *msamCID)
   {
   OSErr err;
   unsigned fileID;
   short setupRef;
   long version = 1;

   err = DoGetIDFromFSSpec(spec, &fileID);

   if (err == noErr)
      {
      err = DoGetSetupDirectoryRefNum(&setupRef);
      }

   if (err == noErr)
      {
      err = DoFindMSAMRecordWithFileID(setupRef, fileID, msamCID);
      }

   /* If you couldn't find the record, create it. */
   if ((err == noErr) && OCEEqualCreationID(msamCID, OCENullCID()))
      {
      err = DoCreateMSAMRecord(setupRef, msamCID);

      if (err == noErr)
         {
```

```
         /* Add the gateway file ID attribute. */
         err = DoAddAttribute(setupRef, msamCID,OCEGetIndAttributeType
                              (kGatewayFileIDAttrTypeNum), sizeof(long),
                              (Ptr) &fileID);
         }

      if (err == noErr)
         {

         /* Add the version attribute. */
         err = DoAddAttribute(setupRef, msamCID, OCEGetIndAttributeType
                              (kVersionAttrTypeNum), sizeof(long),
                              (Ptr) &version);
         }
      }

   return err;
   }

void Initialize()
{
   InitGraf((Ptr) &qd.thePort);
   InitFonts();
   InitWindows();
   InitMenus();
   TEInit();
   InitDialogs(nil);
   InitCursor();
} /* initialize */

main()
   {
   OSErr err;
   StandardFileReply reply;
   CreationID cid;

   Debugger();
   MaxApplZone(); /* expand the heap so that code segments load at the top */

   Initialize();  /* initialize the program */

   StandardGetFile(nil, 0, nil, &reply);
   if (reply.sfGood)
```

```
      {
      err = DoGetMSAMCreationID(&reply.sfFile, &cid);
      Debugger();
      }
   }
```

10. Add a mail service attribute to the MSAM record that points to the Combined record. You can get the record reference, which is a packed record ID, by calling the `kDETcmdGetDSSpec` template callback function.
11. Add a parent MSAM attribute to the Combined record that points to the MSAM record.

    The sample function `DoAddRecordReference` in Listing 4-3 illustrates how to insert a record reference into a record in the Setup catalog.

Listing 4-3 Inserting a record reference into a record

```
OSErr DoAddRecordReference(const CreationID *recordToAddReference,
                  const AttributeType *attrType,
                  const CreationID *referenceCID)
   {
   OSErr err;
   short setupRef;
   RecordID recordReference;
   PackedRecordID *packedReference;
   unsigned short size;

   err = DoGetSetupDirectoryRefNum(&setupRef);

   if (err == noErr)
      {
      recordReference.local.recordName = nil;
      recordReference.local.recordType = nil;
      OCECopyCreationID(referenceCID, &recordReference.local.cid);
      recordReference.rli = nil;

      size = OCEPackedRecordIDSize(&recordReference);
      packedReference = (PackedRecordID*) NewPtr(size);
      err = MemError();
      }

   if (err == noErr)
      {
      OCEPackRecordID(&recordReference, packedReference, size);
```

```
    err = DoAddAttribute(setupRef, recordToAddReference, attrType, size,
                         (Ptr) packedReference);
    }

return err;
}
```

12. Add a standard slot information attribute to the Combined record. This attribute contains a `MailStandardSlotInfoAttribute` structure. See the chapter "Messaging Service Access Modules" in this book for a description of this data structure.
13. Call the `MailCreateMailSlot` function asynchronously to tell the MSAM to create the new slot. You pass this function the MSAM record creation ID, the Combined record creation ID, and some other information. You must call the `kDETcmdBusy` callback routine while waiting for the `MailCreateMailSlot` function to complete. The AOCE system launches the MSAM, which generates a unique slot ID for the new slot and adds it to the Combined record in a slot ID attribute.
14. Delete the "aoce Unconfigured" attribute from the Combined record, because the service is now configured. At this point, the mail and catalog services are available to the user.

Your setup information pages obtain from the user whatever information is required to access the external messaging system, such as the user's account name and password, a telephone number, connection information, and so forth.

Table 4-2 on page 4-65, Table 4-3 on page 4-66, and Table 4-6 on page 4-70 summarize the contents of the MSAM, CSAM, and Combined records.

Adding a Mail Service Only

This section tells you how to add a mail service without adding a catalog service. Read this section if you are providing only an MSAM. It explains what your setup template must do to allow a user to add and configure a mail service.

The setup template for an MSAM includes main aspect templates for a Mail Service record and a Catalog record. Your code resource for the Mail Service record should check whether an MSAM record already exists. If the MSAM record does not exist, your template's code resource must create that record as well.

Listing 4-4 illustrates a setup template for a mail service. Except for changing several strings from "combined" to "mail," using the constant `kMailServiceRecTypeBody` instead of `kCombinedRecTypeBody`, and specifying `notDSAM` instead of `servesDSAM` in the `'sami'` resource, Listing 4-4 is identical to Listing 4-1 on page 4-6.

Listing 4-4 Mail service setup template

```
#define SystemSevenOrLater 1

#include "Types.r"
#include "OCETemplates.h"
#include "OCE.r"

#define kSurfWriterAspect        1000
#define kSurfWriterInfoPage      1250

#define kSignature               'WAVE'
#define kServiceRecordType       kMailServiceRecTypeBody "WAVE"
#define kAspectName              "SurfWriter_MS_Aspect"
#define kInfoPageName            "SurfWriter_MS_Info_Page"

#define kWindowWidth             259
#define kWindowHeight            200
#define kZeroRect                {0, 0, 0, 0}
#define kDoubleLineLeft          kDETSubpageIconLeft
#define kDoubleLineRight         kWindowWidth - kDETSubpageIconLeft
#define kDoubleLineTop           kTopBorder + kFieldHeight + 8
#define kDoubleLineBottom        kDoubleLineTop + 1
#define kDoubleLineRect          {kDoubleLineTop, kDoubleLineLeft,\
                                  kDoubleLineBottom, kDoubleLineRight}


resource 'deta' (kSurfWriterAspect, purgeable)
{
   0, dropCheckConflicts, isMainAspect
};

resource 'rstr' (kSurfWriterAspect + kDETTemplateName, purgeable)
{
   kAspectName
};

resource 'rstr' (kSurfWriterAspect + kDETRecordType, purgeable)
{
   kServiceRecordType
};
```

```
resource 'rstr' (kSurfWriterAspect + kSAMAspectKind, purgeable)
{
   "SurfWriter Mail Service"
};

resource 'rstr' (kSurfWriterAspect + kDETAspectKind, purgeable)
{
   "SurfWriter Mail Service"
};

resource 'rstr' (kSurfWriterAspect + kDETAspectName, purgeable) {
   "unconfigured SurfWriter Mail"
};

resource 'rstr' (kSurfWriterAspect + kSAMAspectUserName, purgeable) {
   "SurfWriter User"
};


resource 'sami' (kSurfWriterAspect + kSAMAspectSlotCreationInfo, purgeable)
{
   2,                                    // max number of catalogs/slots
   kSignature,                           // catalog signature, MSAM type
   servesMSAM,                           // an MSAM template
   notDSAM,                              // not a CSAM template
   "SurfWriter Mail Service ",           // display when user clicks Add
   "untitled SurfWriter Mail Service"    // new record name
};

// Custom window

resource 'detw' (kSurfWriterAspect + kDETAspectInfoPageCustomWindow,
                 purgeable)
   {
   {-1, -1, kWindowHeight, kWindowWidth},
   includePopup
   };

// Include code resource

include "SurfWriterCode" 'detc'(0) as
    'detc'(kSurfWriterAspect+kDETAspectCode, purgeable);
```

```
// Include icons

include "SurfWriterIcons" 'ICN#'(0) as
                  'ICN#'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'icl4'(0) as
                  'icl4'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'icl8'(0) as
                  'icl8'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics#'(0) as
                  'ics#'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics4'(0) as
                  'ics4'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics8'(0) as
                  'ics8'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'SICN'(0) as
                  'SICN'(kSurfWriterAspect+kDETAspectMainBitmap, purgeable);

//------------------------------------------------------------------------
// Info-page

resource 'deti' (kSurfWriterInfoPage, purgeable)
{
   kDefaultSortIndex,
   kZeroRect,
   noSelectFirstText,
   {
      kDETNoProperty, kDETNoProperty, kSurfWriterInfoPage;
   },
   {
   }
};

resource 'rstr' (kSurfWriterInfoPage + kDETTemplateName, purgeable) {
   kInfoPageName
};

resource 'rstr' (kSurfWriterInfoPage + kDETRecordType, purgeable) {
   kServiceRecordType
};

resource 'rstr' (kSurfWriterInfoPage + kDETInfoPageName, purgeable) {
   "SurfWriter Mail Service"
};
```

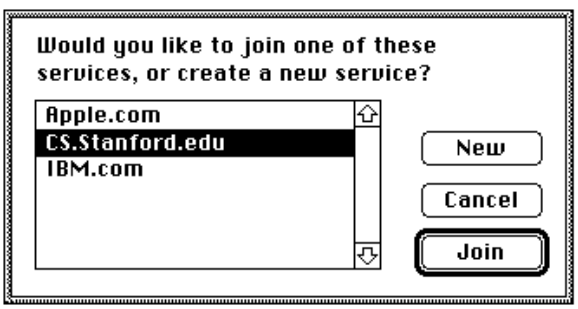
```
resource 'rstr' (kSurfWriterInfoPage + kDETInfoPageMainViewAspect,
                 purgeable) {
   kAspectName
};

resource 'detv' (kSurfWriterInfoPage, "subpageview", purgeable)
   {
   {
   kDoubleLineRect, kDETNoFlags, kDETNoProperty,
   Box { kDETBoxIsGrayed };

   kDETSubpageIconRect, kDETNoFlags, kDETAspectMainBitmap,
   Bitmap { kDETLargeIcon };
   };
   };
```

During system initialization, the Key Chain loads all setup templates. As specified by the `'sami'` resource in Listing 4-4, the Key Chain offers the user the choice "SurfWriter Mail Service" when the user clicks the Add button. When the user selects this choice, the Key Chain scans the Setup catalog looking for catalogs associated with the new mail service. It identifies associated catalogs by looking for Catalog records whose type ends in the catalog signature you specified in the `'sami'` resource of your setup template. In our example, these records would be of type "aoce DirectoryWAVE". If the Key Chain finds any such catalogs that do not already have an associated mail service, it displays a dialog box (Figure 4-1) allowing the user to join one of these existing catalogs.

Figure 4-1 Catalog-choice dialog box



If the user selects one of these catalogs, the Key Chain adds a Mail Service record to the Setup catalog (using the aspect template you provided) and puts an associated catalog attribute in that record pointing to the Catalog record the user selected. The Key Chain also adds to the Catalog record an associated mail service attribute pointing to the Mail Service record it just created.

If the user clicks New or if the Key Chain does not find any associated catalogs that do not already have an associated mail service, it adds to the Setup catalog both a Mail Service record and a new Catalog record. It puts an "aoce Fake" attribute (attribute type index `kFakeAttrTypeNum`) in the Catalog record, indicating that the MSAM does not have a CSAM associated with it. The Key Chain also puts an associated catalog attribute in the Mail Service record and an associated mail service attribute in the Catalog record.

As specified by Listing 4-4, the new Mail Service record's name is "untitled SurfWriter Mail Service", and its type is "aoce Mail ServiceWAVE". The Key Chain writes three additional attributes into this record, as follows:

- the associated catalog attribute
- an attribute of type "aoce Unconfigured" (attribute type index `kUnconfiguredAttrTypeNum`) indicating that the service has not yet been set up
- a version attribute that contains the version number of the Key Chain at the time it created the record

The Key Chain adds a line to the Key Chain window representing the new record. The line includes the key icon used by the Key Chain, the service name you specified in the `kDETAspectName` resource ("unconfigured SurfWriter Mail" in Listing 4-4), and whatever default values your template provides for the Name and Kind fields in the `kSAMAspectUserName` and `kSAMAspectKind` resources ("SurfWriter user" and "SurfWriter Mail Service" in Listing 4-4).

Your code resource should create no new records or attributes at this time. However, when the user opens the Key Chain entry, the Key Chain opens your information page and calls your code resource with the `kDETcmdInstanceInit` routine selector. Your setup template must follow the steps in the following two sections to set up a mail service.

The attributes in the Mail Service record are summarized in Table 4-4 on page 4-67, and the attributes in the Catalog record are summarized in Table 4-5 on page 4-68. Use the Catalog Manager function `DirAddAttributeValue` to add attributes to a record. The function is described in the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces*; see the `DoAddAttribute` function on page 4-18 in Listing 4-2 for an example of its use.

Setting Up the Associated Catalog Service

If the MSAM does not have a CSAM associated with it (that is, if the Catalog record contains an "aoce Fake" attribute), you need to add some configuration information to the Catalog record and rename the record. You can find your Catalog record by unpacking the record reference stored in the associated catalog attribute in your Mail Service record. To add a CSAM and associate it with an existing MSAM, see "Adding a Catalog Service Only" on page 4-28.

1. Write the comment attribute. This is a string that you can use for any purpose. An application developer can read this string by calling the `DirGetExtendedDirectoriesInfo` function; see the chapter "Catalog Manager" in *Inside Macintosh: AOCE Application Interfaces* for a description of this function.

2. Write the real name attribute, containing the external name of the catalog. This name is for your own use; it is not read by the AOCE software. An application developer can read this name by calling the `DirGetExtendedDirectoriesInfo` function. Whereas the catalog name you provide to the `DirSetNameAndType` function (see step 4) must be unique within the AOCE system, the "real" name need not be. For example, the user might have accounts on two different SurfWriter mail servers. Each would have to have a distinct name for display in the Key Chain, but the Catalog records for both could contain the name SurfWriter Mail for the real name attribute.
3. If you wish, you can write a private data attribute. This attribute can contain binary data of any length (up to the maximum length of an attribute) and is for your own use. For example, you can store information about address formats for use by your address template. Your application or template code resource can read this data by calling the `DirGetExtendedDirectoriesInfo` function.
4. Determine the name of the catalog to be used in the Setup catalog, and call the `DirSetNameAndType` function to set the name of the Catalog record to be the same as the catalog name. This name must be unique within the Setup catalog, and once set, it must never be changed.
5. Call the `OCESetupAddDirectoryInfo` function or the `DirAddAttributeValue` function to add the user's record ID attribute to the Catalog record. You must provide the Catalog record's creation ID and a record ID that includes the catalog's name. You can leave blank the other fields in the record ID and the password if the MSAM does not require an account name and password. The `OCESetupAddDirectoryInfo` function is described in the chapter "Authentication Manager" in *Inside Macintosh: AOCE Application Interfaces*.
6. If the name of the User record does not correspond to the user's account name, you can also call the `DirAddAttributeValue` function to add a native name attribute to the Catalog record. This attribute contains the user's name or account name in the external system. This name is for your own use; it is not read by the AOCE software. An application or template code resource can read this name by calling the `OCESetupGetDirectoryInfo` function, described in the chapter "Authentication Manager" in *Inside Macintosh: AOCE Application Interfaces*.
7. Write the discriminator attribute, giving it the value of the address extension type of the messaging system to which your MSAM provides access.

Setting Up the Mail Service

You next need to activate the mail slot. You do this by following the steps described in "Adding the Mail Service" beginning on page 4-12.

Adding a Catalog Service Only

This section tells you how to add a catalog service without adding a mail service. Read this section if you are providing only a CSAM. It explains what your setup template must do to allow a user to add and configure a catalog service.

The setup template for a CSAM includes a main aspect template for a Catalog record.

A basic setup template for a catalog service is identical to the mail service template in Listing 4-4 on page 4-23 with these two exceptions: the word "mail" is replaced with "catalog" throughout, and the following `'sami'` resource is used:

```
resource 'sami' (kSurfWriterAspect + kSAMAspectSlotCreationInfo, purgeable)
{
   2,                                    // max number of catalogs/slots
   kSignature,                           // catalog signature
   notMSAM,                              // not an MSAM template
   servesDSAM,                           // a CSAM template
   "SurfWriter Catalog Service ",        // display when user clicks Add
   "untitled SurfWriter Catalog Service"  // new record name
};
```

During system initialization, the Key Chain loads all setup templates. As specified by the preceding `'sami'` resource, the Key Chain offers the user the choice "SurfWriter Catalog Service" when the user clicks the Add button. When the user selects this choice, the Key Chain scans the Setup catalog looking for unconfigured Catalog records of the type specified by the setup template. In this example, these records would be of type "aoce DirectoryWAVE". It identifies an unconfigured Catalog record by looking for an attribute of type "aoce Fake" in the record. If the Key Chain finds any such Catalog records, it displays a dialog box allowing the user to join one of these existing catalogs (see Figure 4-1 on page 4-26).

If the user selects one of these catalogs, the Key Chain replaces the "aoce Fake" attribute with an "aoce Joined" attribute (attribute type index `kJoinedAttrTypeNum`).

If the user clicks New or if the Key Chain does not find any unconfigured catalogs of the right type, it adds to the Setup catalog a new Catalog record, using the aspect template you provided in your setup template. It puts an "aoce Unconfigured" attribute (attribute type index `kUnconfiguredAttrTypeNum`) in the Catalog record, indicating that it has not yet been configured.

Your code resource should create no new records or attributes at this time. However, when the user opens the Key Chain entry, the Key Chain opens your information page and calls your code resource with the `kDETcmdInstanceInit` routine selector. Your setup template must then set up the catalog as follows:

1. Determine whether you are joining an existing catalog by checking for an "aoce Joined" attribute.
2. If you are joining an existing catalog, the catalog name (which is the record name) is already set and must not be changed. If there is no "aoce Joined" attribute, determine the catalog name and call the `DirSetNameAndType` function to set the name of the Catalog record to be the same as the catalog name. This name must be unique within the Setup catalog, and once set, it must never be changed.
3. If you are joining an existing catalog, check the attributes in the Catalog record (see Table 4-5 on page 4-68) for existing information.

4. Preserve any existing information and follow the steps in "Adding the Catalog Service" beginning on page 4-10 as appropriate to provide any attributes that don't already exist.
5. Delete the "aoce Unconfigured" or "aoce Joined" attribute.

Modifying an Existing Service

Each time the user restarts his or her system, the Key Chain calls your code resource with the `kDETcmdInit` routine selector. At that time you should obtain the file system specifier (`FSSpec` structure) for your MSAM file and make sure that the file ID is the same as that saved in the gateway file ID attribute in the MSAM record (Table 4-2 on page 4-65). If the user has replaced the MSAM record, the file ID for your MSAM file will not match the one saved in the gateway file ID attribute. In this case, you must replace the gateway file ID attribute with the correct file ID. Because the Collaboration toolbox reads the file IDs from the MSAM record before calling your code resource, it cannot open the new MSAM file until the user restarts the system. Therefore, after updating the MSAM record, your code resource must display a dialog box telling the user to restart the system in order to use the new file.

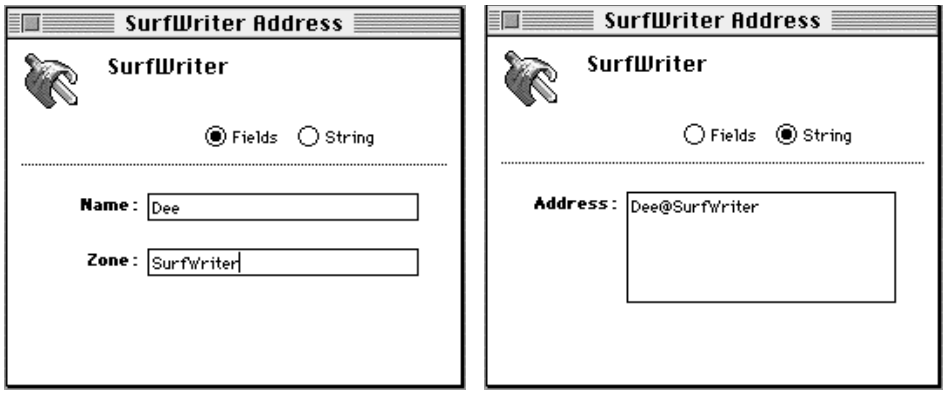
Your setup template must provide one or more information pages that allow a user to modify an existing service. The implementation of this feature is up to you.

Writing and Modifying Addresses

To allow users to add addresses of the type used by your MSAM to their User records, you must provide an address template. An address template consists of a main aspect template for addresses of the type used by your mail service plus at least one information page. All addresses used by AOCE are in the format of an `OCEPackedRecipient` data structure, as described in the chapter "Messaging Service Access Modules" in this book.

Whenever practical, an address information page should provide two alternative but equivalent methods of entering addresses: a set of fields and a single input string. Internally, the form of the address is defined by the MSAM providing the address template. Figure 4-2 shows the two views of an address template for the SurfWriter application.

Figure 4-2 Alternate forms of a single address information page



Writing an Address Template

Listing 4-5 shows the AOCE aspect and information page templates that create the information page shown in Figure 4-2. The aspect template defines an attribute of type `kMailSlotsAttrTypeBody`. The Collaboration toolbox expects all addresses to be stored in a multivalued attribute of that type. The attribute tag specifies the address format. In this example, the attribute tag is `'WAVE'`, the signature of the fictional application SurfWriter. The lookup table lists the properties that must be processed by the code resource and uses a custom lookup-table element so that the CE calls the code resource each time it processes the lookup table.

The information page in Listing 4-5 has two conditional views to provide the two methods of entering addresses: Fields or String. One view or the other is displayed, depending on which radio button the user clicked.

Because an address is of type `OCEPackedRecipient`, which is a packed, private data structure, you must use a code resource to pack and unpack this structure. The code resource can call the utility routines described in the chapter "Interprogram Messaging Manager" in *Inside Macintosh: AOCE Application Interfaces* to pack and unpack `OCEPackedRecipient` structures. The code resource also converts between the fields and string forms of the address. Listing 4-6 on page 4-41 shows the code resource for the address template in Listing 4-5.

Listing 4-5 Address template

```
// Defines for the headers of the templates

#define kZeroRect {0, 0, 0, 0}

#define kWindowWidth 259
#define kWindowHeight 200
```

```
#define kDETMenuLeft kDETSubpageIconRight + 16  // left edge of subpage menu
#define kDETMenuRight kDETMenuLeft + 150        // right edge of subpage menu
#define kDETMenuBottom kDETSubpageIconTop + 22  // bottom of subpage menu

#define kTopBorder kDETSubpageIconBottom + 8    // top of first item
#define kFieldHeight 16                         // height of fields
#define kMenuWidth 75                           // width of "View as" menu
#define kMenuTitleWidth 60               // width allowed for field titles
#define kFieldTitleSeparator 5

// These defines are for the "View As:" radio buttons at the top of
// most of the address templates.

#define kViewAsTextWidth 50
#define kViewButton1Width 43
#define kViewButton2Width 45
#define kFirstFieldTop kHeaderBottom + 10

#define kViewAsTextTop kTopBorder + 1
#define kViewAsTextLeft kDETMenuLeft
#define kViewAsTextBottom kViewAsTextTop + 14
#define kViewAsTextRight kViewAsTextLeft + kViewAsTextWidth
#define kViewAsTextRect {kViewAsTextTop, kViewAsTextLeft, kViewAsTextBottom,
                         kViewAsTextRight}

#define kFieldsButtonTop kTopBorder
#define kFieldsButtonBottom kFieldsButtonTop + kFieldHeight
#define kFieldsButtonLeft kViewAsTextRight + kFieldTitleSeparator
#define kFieldsButtonRight kFieldsButtonLeft + kViewButton1Width
#define kFieldsButtonRect {kFieldsButtonTop, kFieldsButtonLeft,
                           kFieldsButtonBottom, kFieldsButtonRight}

#define kStringButtonTop kTopBorder
#define kStringButtonBottom kStringButtonTop + kFieldHeight
#define kStringButtonLeft kFieldsButtonRight + kFieldTitleSeparator +
                          kFieldTitleSeparator
#define kStringButtonRight kStringButtonLeft + kViewButton2Width
#define kStringButtonRect {kStringButtonTop, kStringButtonLeft,
                           kStringButtonBottom, kStringButtonRight}

// These defines are for miscellaneous components, such as the dotted line
at // the top of the template.
```

```
#define kDoubleLineLeft kDETSubpageIconLeft
#define kDoubleLineRight kWindowWidth - kDETSubpageIconLeft
#define kDoubleLineTop kTopBorder + kFieldHeight + 8
#define kDoubleLineBottom kDoubleLineTop + 1
#define kDoubleLineRect {kDoubleLineTop, kDoubleLineLeft, kDoubleLineBottom,
                         kDoubleLineRight}

#define kHeaderBottom kDoubleLineBottom

#define kLeftBorder 35                  // left border
#define kTitleWidth 45                  // width allowed for field titles
#define kFieldLeft kLeftBorder + kTitleWidth
                                        // left border of field items
#define kTextLeft 10                    // left border of static text items
#define kTextRight kFieldLeft - kFieldTitleSeparator
                                        // right border of static text items
#define kFieldWidth 155                 // width of text fields

// Error messages for use by the code resource
resource 'STR#' (kSurfInfoPageAspect, purgeable) {
   {
   /* [1] */   "An unspecified problem occurred.",
   /* [2] */   "This address must contain a name. Please enter a name in "
               "the appropriate field.",
   }
};

// Properties

#define prMyName (kDETFirstDevProperty + 0)
#define prMyZone (kDETFirstDevProperty + 1)
#define prMyAddress (kDETFirstDevProperty + 2)
#define prViewMenu (kDETFirstDevProperty + 3)
#define prInited (kDETFirstDevProperty + 4)
#define prOldName (kDETFirstDevProperty + 5)
#define prOldZone (kDETFirstDevProperty + 6)
#define prOldAddress (kDETFirstDevProperty + 7)
#define prDefaultName (kDETFirstDevProperty+8)
#define prDefaultZone (kDETFirstDevProperty+9)
#define prDefaultDisplayName (kDETFirstDevProperty+10)
```

```
//-------------------------------------------------------------------------
// These custom lookup-table elements cause the Catalogs Extension to call
// the code resource.

#define kProcessData 'Hey!'
#define kPostProcessData 'Done'

#define kNullNameError 1000

//-------------------------------------------------------------------------*/
// Aspect template
include "SurfAddressCode" 'detc'(0) as 'detc'(kSurfInfoPageAspect +
    kDETAspectCode, purgeable);

// Aspect template signature resource
resource 'deta' (kSurfInfoPageAspect, purgeable) {
   1000,                // drop-operation order
   dropCheckConflicts,  // drop check flag
   isMainAspect         // is the main aspect
};

// Template name
resource 'rstr' (kSurfInfoPageAspect + kDETTemplateName, purgeable) {
   kNewSurfAddressAspectName
};

// This template applies to the kMailSlotsAttrTypeBody attribute
resource 'rstr' (kSurfInfoPageAspect + kDETAttributeType, purgeable) {
   kMailSlotsAttrTypeBody
};

// Tag of attribute this applies to
resource 'detn' (kSurfInfoPageAspect + kDETAttributeValueTag){
   'WAVE'
};

// Template kind
resource 'rstr' (kSurfInfoPageAspect + kDETAspectKind, purgeable) {
   "SurfWriter mail address"
};

// String for Add dialog box
resource 'rstr' (kSurfInfoPageAspect + kDETAspectNewMenuName, purgeable) {
```

```
   "SurfWriter"
};

// Category for template
resource 'rst#' (kSurfInfoPageAspect+kDETAspectCategory,purgeable)
{
   {
      kDETCategoryAddressItems,
   }
};

// Open info page automatically when user creates new attribute value.
resource 'detn' (kSurfInfoPageAspect + kDETAspectSublistOpenOnNew){
   1
};

// Attribute tag followed by default new value for attribute. This resource
// must be present if user is allowed to add a new attribute. The code
// resource routine DoCreateNewAttribute (page 4-44) appends the default
// attribute value to the attribute tag to create the new attribute value.
data 'detb' (kSurfInfoPageAspect + kDETAspectNewValue, purgeable) {
   $"5741 5645"              // 'WAVE' tag
};

// Lookup table. Most property values are actually set by the code resource,
// but it is necessary to include them all here so the Catalogs Extension
// to the Finder will know they exist and so that it will call the code
// resource when the user changes their values.
resource 'dett' (kSurfInfoPageAspect + kDETAspectLookup, purgeable)
{{
   { kMailSlotsAttrTypeBody },
   'WAVE',
   useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
      kProcessData,kDETNoProperty,  0; // custom property element,
                                       //  causing CE to call code resource
      'prop', prMyZone, 0;             // declare all the properties
      'prop', prMyName, 0;
      'prop', prMyAddress, 0;
      'prop', kDETAspectName, 0;
```

```
      'awrd',  kDETNoProperty, 0;      // align to a word boundary
      kPostProcessData, kDETNoProperty, 0;
                                       // post-process the data
                                          (if necessary)
      'Pref',  kDETNoProperty, 0;      // ask CE to save preferred
                                       // mailslot info
      };
}};

// Custom window information
resource 'detw' (kSurfInfoPageAspect + kDETAspectInfoPageCustomWindow,
purgeable)
   {
   {0, 0, kWindowHeight, kWindowWidth},
   includePopup         // this places label in info page
   };


// Default values for some properties

resource 'detn' (kSurfInfoPageAspect + prViewMenu, purgeable) {
   1           // make "fields" the default conditional view
};

resource 'rstr'(kSurfInfoPageAspect + prDefaultName, purgeable) {
   "<Name>"
};
resource 'rstr'(kSurfInfoPageAspect + prDefaultZone, purgeable) {
   "<Zone>"
};
resource 'rstr'(kSurfInfoPageAspect + prDefaultDisplayName, purgeable) {
   "untitled SurfWriter address"
};

// Text for help balloons

// Text for help balloon for the item in a sublist
resource 'rstr' (kSurfInfoPageAspect+kDETAspectWhatIs, purgeable) {
   "Contains a SurfWriter mail address."
};

// Text for help balloon for an alias to the item
resource 'rstr' (kSurfInfoPageAspect+kDETAspectAliasWhatIs, purgeable) {
```

```
   "Contains an alias to a SurfWriter mail address."
};

// Text for help balloons for the properties
resource 'rst#' (kSurfInfoPageAspect+kDETAspectBalloons, purgeable) {
   {
   "Shows the name for this SurfWriter mail address. You can edit this "
      "name.",
   "Shows the name for this SurfWriter mail address. You cannot edit this "
      "name.",
   "Shows the zone location for this SurfWriter mail address. You can edit "
      "this zone name.",
   "Shows the zone location for this SurfWriter mail address. You cannot "
      "edit this zone name.",
   "Shows this SurfWriter mail address as a series of characters. "
      "You can edit this address.",
   "Shows this SurfWriter mail address as a series of characters. "
      "You cannot edit this address.",
   "Controls the display of address information. Click a button to change "
      "the display.",
   "",
   }
};


//-----------------------------------------------------------------------------
// Information page
//
#define kTwoProperty kDETFirstConstantProperty + 2

// Information page signature resource
resource 'deti' (kSurfInfoPage, purgeable) {
   1000,               // sort order
   kZeroRect,          // no sublist
   noSelectFirstText,  // don't select first text field
   {
   kDETNoProperty, kDETNoProperty, kSurfInfoPage;  // common view list
   prViewMenu, kDETOneProperty, kSurfInfoPage + 1; // "fields" view list
   prViewMenu, kTwoProperty, kSurfInfoPage + 2;    // "strings" view list
   },
   {                   // no subview view lists
   }};
```

```
// Template name
resource 'rstr' (kSurfInfoPage + kDETTemplateName, purgeable) {
   kNewSurfAddressInfoPageName
};

// Attribute type
resource 'rstr' (kSurfInfoPage + kDETAttributeType, purgeable) {
   kMailSlotsAttrTypeBody
};

// Name of information page that appears as label on page
resource 'rstr' (kSurfInfoPage + kDETInfoPageName, purgeable) {
   "SurfWriter"
};

// Name of related aspect template
resource 'rstr' (kSurfInfoPage + kDETInfoPageMainViewAspect, purgeable) {
   kNewSurfAddressAspectName
};

// Defines for the rest of the info page

#define kAddressWidth 155          // width of string address field
#define kAddressHeight 64          // height of string address field

#define kStatText1Top kHeaderBottom + kFieldHeight
#define kStatText1Left kTextLeft
#define kStatText1Bottom kStatText1Top + kFieldHeight
#define kStatText1Right kTextRight
#define kStatText1Rect {kStatText1Top, kStatText1Left, kStatText1Bottom,
kStatText1Right}

#define kNameTextTop kStatText1Top
#define kNameTextLeft kFieldLeft
#define kNameTextBottom kStatText1Bottom
#define kNameTextRight kNameTextLeft + kFieldWidth
#define kNameTextRect {kNameTextTop, kNameTextLeft, kNameTextBottom,
kNameTextRight}

#define kStatText2Top kStatText1Bottom + kFieldHeight
#define kStatText2Left kTextLeft
#define kStatText2Bottom kStatText2Top + kFieldHeight
#define kStatText2Right kTextRight
```

```
#define kStatText2Rect {kStatText2Top, kStatText2Left, kStatText2Bottom,
kStatText2Right}

#define kZoneTextTop kStatText2Top
#define kZoneTextLeft kFieldLeft
#define kZoneTextBottom kStatText2Bottom
#define kZoneTextRight kZoneTextLeft + kFieldWidth
#define kZoneTextRect {kZoneTextTop, kZoneTextLeft, kZoneTextBottom,
kZoneTextRight}

#define kAddressTextTop kNameTextTop
#define kAddressTextLeft kNameTextLeft
#define kAddressTextBottom kAddressTextTop + kAddressHeight
#define kAddressTextRight kAddressTextLeft + kAddressWidth
#define kAddressTextRect {kAddressTextTop, kAddressTextLeft,
kAddressTextBottom, kAddressTextRight}

// Nonconditional view
resource 'detv' (kSurfInfoPage, purgeable)
   {
   {
   kDoubleLineRect, kDETNoFlags, kDETNoProperty,
   Box { kDETBoxIsGrayed };

   kDETSubpageIconRect, kDETNoFlags, kDETAspectMainBitmap,
   Bitmap { kDETLargeIcon };

   kViewAsTextRect, kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
                        kDETLeft, kDETBold, "View as:" };

   kFieldsButtonRect, kDETEnabled, prViewMenu,
   RadioButton { kDETApplicationFont, kDETApplicationFontSize,
                 kDETLeft, kDETNormal, "Fields", prViewMenu, 1 };

   kStringButtonRect, kDETEnabled, prViewMenu,
   RadioButton { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
                 kDETNormal, "String", prViewMenu, 2 };
   };
   };

// "Fields" conditional view
resource 'detv' (kSurfInfoPage + 1, purgeable)
```

```
   {
   {
   kStatText1Rect, kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
                        kDETRight, kDETBold, "Name:" };

   kStatText2Rect, kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
                        kDETRight, kDETBold, "Zone:" };

   kNameTextRect, kDETNoFlags, prMyName,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
              kDETNormal };

   kZoneTextRect, kDETNoFlags, prMyZone,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
              kDETNormal };
   };
   };

// "String" conditional view
resource 'detv' (kSurfInfoPage + 2, purgeable)
   {
   {
   kStatText1Rect, kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
kDETRight,
                  kDETBold, "Address:" };

   kAddressTextRect, kDETMultiLine, prMyAddress,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
kDETNormal };
   };
   };

//
// Icons

include "AlbumIcons" 'ICN#'(0) as
      'ICN#'(kSurfInfoPageAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'icl4'(0) as
      'icl4'(kSurfInfoPageAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'icl8'(0) as
      'icl8'(kSurfInfoPageAspect + kDETAspectMainBitmap, purgeable);
```

```
include "AlbumIcons" 'ics#'(0) as
      'ics#'(kSurfInfoPageAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'ics4'(0) as
      'ics4'(kSurfInfoPageAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'ics8'(0) as
      'ics8'(kSurfInfoPageAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'SICN'(0) as
      'SICN'(kSurfInfoPageAspect + kDETAspectMainBitmap, purgeable);
```

Writing an Address Template Code Resource

The code resource for the address template shown in Listing 4-5 consists of a dispatcher routine and several routines to handle standard requirements of the CE, as described in the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces*. The dispatcher routine and main code resource routines are shown in the following section. The sections "Data Input Subroutines for the Address Template" beginning on page 4-47, "Data Output Subroutines for the Address Template" beginning on page 4-51, and "Miscellaneous Subroutines" beginning on page 4-57 show the subroutines called by the main routines and by other subroutines. Subroutines with names beginning with `My` (for example, `MyCreateFieldsFromOCERecipient`) are described but not shown here.

Main Routines for the Address Template Code Resource

Listing 4-6 shows the main routines for the code resource. The `DoSurfAddress` routine is the dispatcher called by the Catalogs Extension (CE). When the CE calls this routine, it calls one of the other routines shown in this section, depending on the routine selector passed by the CE.

Listing 4-6 Main routines of the address template code resource

```
/* Code resource main routine */
pascal OSErr DoSurfAddress(DETCallBlockPtr callBlockPtr)
{
   OSErr err = 1;
/* Process only calls targeted to this template plus untargeted calls. */
   if ((callBlockPtr->protoCall.reqFunction < kDETcmdTargetedCall) ||
(callBlockPtr->protoCall.target.selector == kDETSelf))
   {
      switch (callBlockPtr->protoCall.reqFunction)
      {
         case kDETcmdInit:
            err = DoInitTemplate(callBlockPtr);
            break;
```

```
            case kDETcmdAttributeCreation:
               err = DoCreateNewAttribute(callBlockPtr);
               break;

            case kDETcmdInstanceInit:
               err = DoInitInstance(callBlockPtr);
               break;

            case kDETcmdInstanceExit:
               err = DoExitInstance(callBlockPtr);
               break;

            case kDETcmdPropertyDirtied:
               err = DoPropertyDirty(callBlockPtr);
               break;

            case kDETcmdValidateSave:
               err = DoPrepareToSave(callBlockPtr);
               break;

            case kDETcmdPatternIn:
               err = DoPatternIn(callBlockPtr);
               break;

            case kDETcmdPatternOut:
               err = DoPatternOut(callBlockPtr);
               break;

            default:
               break;
         }
      }

      return err;
}

/* ------------------------------------------------------------------------ */
/* Template initialization. Set call-for mask. */
static OSErr DoInitTemplate(DETCallBlockPtr callBlockPtr)
{
   OSErr err = noErr;
   callBlockPtr->init.newCallFors = kDETCallForValidation +
       kDETCallForAttributes + kDETCallForViewChanges;
```

```
   return err;
}

/* ------------------------------------------------------------------------ */
/* Aspect initialization */
static OSErr DoInitInstance(DETCallBlockPtr callBlockPtr)
{
   OSErr err = noErr;
/* Set value of property prInited to false. */
   DoSetInited(callBlockPtr, false);

   return err;
}

/* ------------------------------------------------------------------------ */
/* Exit routine */
static OSErr DoExitInstance(DETCallBlockPtr callBlockPtr)
{
   OSErr err = noErr;
/* Set value of property prInited to false. */
   DoSetInited(callBlockPtr, false);

   return err;
}

/* ------------------------------------------------------------------------ */
/* The CE is about to save property values. */
static OSErr DoPrepareToSave(DETCallBlockPtr callBlockPtr)
{
   OSErr err = kDETDidNotHandle;
   Handle errorStr = nil;
   if (!DoIsInited(callBlockPtr))
      return 1;

/* Check the data to make sure it's valid. */
   err = MyCheckData(callBlockPtr);
   return err;
}

/* ------------------------------------------------------------------------ */
/* The CE calls this routine to process the custom lookup-table element that
   processes input data. This routine is discussed in
```

```
   "Data Input Subroutines for the Address Template" beginning on
   page 4-47.*/

static OSErr DoPatternIn(DETCallBlockPtr callBlockPtr)
{
   OSErr err = 1;
   Boolean enabled = true;
   /* For input processing only */
   if (callBlockPtr->patternIn.elementType == kProcessData)
   {
      err = DoExtractInformation(callBlockPtr);
   }
   else if (callBlockPtr->patternIn.elementType == kPostProcessData)
   {
      err = DoSetDisplayName(callBlockPtr);
   }

   DoHandleError(callBlockPtr, err);

   return err;
}

/* ------------------------------------------------------------------------ */
/* The CE calls this routine when it is about to add a new attribute value
   to the sublist. This routine gets the default values for the field
   properties and the default string to be displayed in the sublist for the
   attribute value. These values are provided by the template (page 4-36).
   It packs the fields into a string and appends them to the default value
   of the attribute. The CE provides a pointer to this default attribute
   value; it gets the default attribute value from the kDETAspectNewValue
   resource in the aspect template (page 4-35). See
   beginning on page 4-51 for a discussion of the DoWriteNameAndZone
   subroutine. */

static OSErr DoCreateNewAttribute(DETCallBlockPtr callBlockPtr)
{
   OSErr err = 1;
   RStringPtr *name, *zone, *dName;
   /* Initialize the data. */
   name = nil;
   zone = nil;
   dName = nil;
```

```
   /* Get the name. */
   name = (RStringPtr*) GetResource('rstr', kSurfInfoPageAspect +
                                    prDefaultName);

   /* Get the zone. */
   zone = (RStringPtr*) GetResource('rstr', kSurfInfoPageAspect +
                                    prDefaultZone);

   /* Get the display name. */
   dName = (RStringPtr*) GetResource('rstr', kSurfInfoPageAspect +
                                     prDefaultDisplayName);

   /* Lock everything. */
   HLock((Handle) name);
   HLock((Handle) zone);
   HLock((Handle) dName);

   /* Write out the data. */
   err = DoWriteNameAndZone(callBlockPtr, *name, *dName, *zone,
                            callBlockPtr->attributeCreationBlock.value);

   /* Unlock everything. */
   HUnlock((Handle) name);
   HUnlock((Handle) zone);
   HUnlock((Handle) dName);

   /* Clean up. */
   ReleaseResource((Handle) name);
   ReleaseResource((Handle) zone);
   ReleaseResource((Handle) dName);

   if (err == noErr)
      err = kDETDidNotHandle;

   return err;
}

/* ------------------------------------------------------------------------ */
/* The Catalogs Extension calls this routine when you call the
    kDETcmdDirtyProperty callback routine to indicate that a property value
    has changed, requiring a view to be redrawn. The CE also calls this
    routine when the CE completes its first catalog lookup. If the user has
    changed either of the text fields in the "Fields" view of the info page,
```

```
    this routine updates the value of the address string. If the user has
    changed the address in the "String" view of the info page, this routine
    updates the values of the address fields. The DoUpdateNameAndZone and
    DoUpdateAddress functions are shown in
    “Data Output Subroutines for the Address Template” beginning on
    page 4-51. */

static OSErr DoPropertyDirty(DETCallBlockPtr callBlockPtr)
{
   OSErr err = noErr;
   short dirtyProperty;
   if (!DoIsInited(callBlockPtr))
      return 1;

   dirtyProperty = callBlockPtr->propertyDirtied.property;

   if (dirtyProperty == prMyAddress)
   {
      err = DoUpdateNameAndZone(callBlockPtr);
   }
   else if ((dirtyProperty == prMyName) || (dirtyProperty == prMyZone))
   {
      err = DoUpdateAddress(callBlockPtr);
   }

   if (err == noErr)
   {
      err = 1;
   }

   DoHandleError(callBlockPtr, err);

   return err;
}

/* ------------------------------------------------------------------------ */
/* The CE calls this routine when processing the custom lookup-table
    element that processes output data. This routine is discussed in
    “Data Output Subroutines for the Address Template” on page 4-51.*/
```

```
static OSErr DoPatternOut(DETCallBlockPtr callBlockPtr)
{
   OSErr err = 1;
   long message;
   message = callBlockPtr->patternIn.elementType;

   if (!DoIsInited(callBlockPtr))
      return 1;

   /* Write data only if you're supposed to be writing */
   if (message == kProcessData)
   {
      err = DoWriteData(callBlockPtr);
   }

   return err;
}
```

Data Input Subroutines for the Address Template

The CE calls your code resource's `DETcmdPatternIn` routine when it has to process a custom lookup-table element for input data. The `DoPatternIn` function shown in Listing 4-6 calls the `DoExtractInformation` function shown in Listing 4-7. The `DoExtractInformation` function calls the `DoReadData` function, which in turn calls the `MyCreateFieldsFromOCERecipient` function to read the address fields from the `OCERecipient` structure. Then the `DoReadData` function calls the `MyCreateAddressString` function, which creates an address string from the address fields. Finally, the `DoReadData` function calls the `DoSetAllStringProperties` function, which sets the values of the field and string properties for display in the information page.

Next, the `DoExtractInformation` function calls the `DoExtractDisplayName` function, passing it the attribute data; that is, the address in the form of an `OCEPackedRecipient` structure. The `DoExtractDisplayName` function unpacks the `OCEPackedRecipient` structure and extracts the record name. It sets the name of the attribute value to the record name. The Electronic Addresses information page uses this name for display in the sublist.

Finally, the `DoExtractInformation` function calculates the size of the data it just read and sets the data offset and bit offset fields to the end of the attribute data so that the CE stops processing the attribute.

The `DoPatternIn` function (page 4-44) also calls the `DoSetDisplayName` function (page 4-51). The `DoSetDisplayName` function checks whether the attribute value has been assigned a display name (that is, a name to display for the attribute value in the sublist or for a stand-alone attribute). If not, it sets the display name to be the same as the string in the property `prMyName`. That property is also used for the Name field in the Fields address information page (see Figure 4-2 on page 4-31).

Listing 4-7 Input subroutines for the address template code resource

```
static OSErr DoExtractInformation(DETCallBlockPtr callBlockPtr)
{

   OSErr err = 1;
   PackedDSSpecPtr pds;
   Boolean enabled = true;
   short size;
   pds = (PackedDSSpecPtr) callBlockPtr->patternIn.attribute->value.bytes;
   if (DoGetXtnType(pds) == 'WAVE')
   {
      /* Read the data in */
      err = DoReadData(callBlockPtr, pds);

      err = DoExtractDisplayName(callBlockPtr, pds);

      size = (* ((short *) callBlockPtr->patternIn.attribute->value.bytes))
               + 2;
      callBlockPtr->patternIn.dataOffset = size;
      callBlockPtr->patternIn.bitOffset = 0;
   }
   else
   {
      /* This isn't a SurfWriter address.  Abort! */
      enabled = false;
   }

   if ((err == noErr) && enabled)
   {
      DoSetInited(callBlockPtr, true);
   }

   return err;
}

/* ------------------------------------------------------------------------ */

static long DoGetXtnType(PackedDSSpecPtr pds)
{
   long extensionType;
   DSSpec spec;
   RecordID rid;
```

```
   OCEUnpackDSSpec(pds, &spec, &rid);
   extensionType = spec.extensionType;

   return extensionType;
}
/* ------------------------------------------------------------------------ */

static OSErr DoReadData(DETCallBlockPtr callBlockPtr, PackedDSSpecPtr pds)
{
   OSErr err = noErr;
   RStringPtr name, zone, address;
   /* Initialize data */
   name = nil;
   zone = nil;
   address = nil;

   /* Extract the info we want */
   err = MyCreateFieldsFromOCERecipient(pds, &name, &zone);

   if (err == noErr)
   {
      /* Create the address string for display */
      err = MyCreateAddressString(name, zone, &address);
   }

   if (err == noErr)
   {
      err = DoSetAllStringProperties(callBlockPtr, name, zone, address);
   }

   DisposeIfPtr(name);
   DisposeIfPtr(zone);
   DisposeIfPtr(address);

   return err;
}
/* ------------------------------------------------------------------------ */

static OSErr DoSetAllStringProperties(DETCallBlockPtr callBlockPtr,
                                      RStringPtr name,
                                      RStringPtr zone,
                                      RStringPtr address)
```

```
{
   OSErr err = noErr;
   /* Set the zone property. */
   err = DoSetRStringProperty(callBlockPtr, prMyZone, zone, false);

   /* Set the name property for the list view. */
   err = DoSetRStringProperty(callBlockPtr, prMyName, name, false);

   /* Set the OldName property for the list view. */
   err = DoSetRStringProperty(callBlockPtr, prOldName, name, false);

   /* Create the address for display. */
   err = DoSetRStringProperty(callBlockPtr, prMyAddress, address, false);

   /* Create the OldAddress for display. */
   err = DoSetRStringProperty(callBlockPtr, prOldAddress, address, false);
   return err;
}
/* ------------------------------------------------------------------------ */

static OSErr DoExtractDisplayName(DETCallBlockPtr callBlockPtr,
PackedDSSpecPtr pds)
{
   OSErr err = noErr;
   DSSpec spec;
   RecordID rid;
   RStringPtr dName;

   OCEUnpackDSSpec(pds, &spec, &rid);
   dName = rid.local.recordName;

   if (dName != nil)
   {
      err = DoSetRStringProperty(callBlockPtr, kDETAspectName, dName, false);
      if (err == kDETPropertyBusy)
      {
         err = noErr;
      }
   }

   return err;
}
```

```
/* ------------------------------------------------------------------------ */

static OSErr DoSetDisplayName(DETCallBlockPtr callBlockPtr)
{
   OSErr err = noErr;
   RStringPtr name, dName;

   name = nil;
   dName = nil;

   /* Get the name. */
   err = DoGetRStringPtrProperty(callBlockPtr, prMyName, &name);

   /* Get the display name. */
   err = DoGetRStringPtrProperty(callBlockPtr, kDETAspectName, &dName);

   if (dName->dataLength <= 0)
   {
      err = DoSetRStringProperty(callBlockPtr, kDETAspectName, name, false);
   }

   DisposeIfPtr(name);
   DisposeIfPtr(dName);

   return err;
}
```

Data Output Subroutines for the Address Template

The CE calls your code resource's `DETcmdPatternOut` routine when it has to process a custom lookup-table element for output data. The parameter block that the CE provides with the `DETcmdPatternOut` call to your code resource includes a handle to the attribute. The attribute already has an attribute tag assigned but lacks the data length and data fields. The `DoPatternOut` function shown in Listing 4-6 calls the `DoWriteData` function shown in Listing 4-8, reads the address fields from the current property values, and calls the `DoWriteNameAndZone` function. The `DoWriteNameAndZone` function calls the `DoPackNameAndZone` function, which verifies the name and zone and calls the `MyCreateOCERecipient` function. The `MyCreateOCERecipient` function packs the address fields into a string and creates an `OCEPackedRecipient` structure that includes the address string as an extension. Then the `DoWriteNameAndZone` function appends the `OCEPackedRecipient` structure to the data handle provided by the CE, thus updating the attribute value. The `MyCreateOCERecipient` function is not shown here. For a sample function that creates an `OCEPackedRecipient` structure for an external messaging system, see "Translating to an AOCE Address" beginning on page 2-88 in the chapter "Messaging Service Access Modules" in this book.

The Catalogs Extension calls your `kDETcmdPropertyDirtied` routine when a property value has changed, requiring a view to be redrawn. If the user has changed either of the text fields in the "Fields" view of the info page, the `DoPropertyDirty` function shown in Listing 4-6 calls the `DoUpdateAddress` function to update the value of the address string. If the user has changed the address in the "String" view of the information page, the `DoPropertyDirty` function calls the `DoUpdateNameAndZone` function to update the values of the address fields.

Listing 4-8 Output subroutines for the address template code resource

```
static OSErr DoWriteData(DETCallBlockPtr callBlockPtr)
{
   OSErr err = noErr;
   RStringPtr name, zone, dName;
   Size size;
   /* Initialize the data. */
   name = nil;
   zone = nil;
   dName = nil;

   /* Get the name. */
   err = DoGetRStringPtrProperty(callBlockPtr, prMyName, &name);

   /* Get the zone. */
   if (err == noErr)
   {
      err = DoGetRStringPtrProperty(callBlockPtr, prMyZone, &zone);
   }

   /* Get the display name. */
   if (err == noErr)
   {
      err = DoGetRStringPtrProperty(callBlockPtr, kDETAspectName, &dName);
   }

   /* Write out the data. */
   if (err == noErr)
   {
      err = DoWriteNameAndZone(callBlockPtr, name, dName, zone,
                               callBlockPtr->patternOut.data);
   }
```

```
   if (err == noErr)
   {
      size = GetHandleSize((Handle) callBlockPtr->patternOut.data);
      callBlockPtr->patternOut.dataOffset = size;
      callBlockPtr->patternOut.bitOffset = 0;
   }

   DisposeIfPtr(name);
   DisposeIfPtr(zone);
   DisposeIfPtr(dName);

   return err;
}

/* ------------------------------------------------------------------------ */

static OSErr DoWriteNameAndZone(DETCallBlockPtr callBlockPtr, RStringPtr
name, RStringPtr dName, RStringPtr zone, Handle buffer)
{
   OSErr err = noErr;
   PackedDSSpecPtr pds = nil;
   Size size;

   /* Pack the data. */
   err = DoPackNameAndZone(callBlockPtr, name, dName, zone, &pds);

   /* Append the address to the data handle provided by the CE. */
   if (err == noErr)
   {
      size = GetPtrSize((Ptr) pds);

      SetHandleSize(buffer, size);
      err = MemError();

      if (err == noErr)
         BlockMove((Ptr) pds, *buffer, size);
   }

   /* Dispose of allocated stuff. */
   DisposeIfPtr(pds);

   return err;
}
```

```
/* ------------------------------------------------------------------------ */
static OSErr DoPackNameAndZone(DETCallBlockPtr callBlockPtr, RStringPtr
name, RStringPtr dName, RStringPtr zone, PackedDSSpecPtr* pds)
{
   OSErr err = noErr;
   unsigned short size;
   /* Initialize the data. */
   *pds = nil;

   /* Validate the name and zone. */
   if (!DoStringPtrIsOK(name) || !DoStringPtrIsOK(zone))
      err = paramErr;

   /* Create an address. */
   if (err == noErr)
   {
      err = MyCreateOCERecipient(pds, &size, name, dName, zone);
   }

   if ((*pds == nil) && (err == noErr))
      err = paramErr;

   return err;
}
/* ------------------------------------------------------------------------ */
/* This routine is called to make sure that the name and zone are
    not of zero length and that they have valid values. */

static Boolean DoStringPtrIsOK(RStringPtr string)
{
   Boolean good = true;
   Str255 divider;
   if (string->dataLength < 0)
      good = false;

   if (string->dataLength > 0)
   {
      if (MyValidateString(string, divider, 0) != kBadString)
         good = false;
   }

   return good;
}
```

```
/* ------------------------------------------------------------------------ */

static OSErr DoUpdateNameAndZone(DETCallBlockPtr callBlockPtr)
{
   OSErr err = noErr;
   OSErr tempErr;
   RStringPtr name, zone, address, oldAddress;
   /* Initialize the variables. */
   name = nil;
   zone = nil;
   address = nil;
   oldAddress = nil;

   err = DoGetRStringPtrProperty(callBlockPtr, prMyAddress, &address);

   if (err == noErr)
   {
      err = MyDecomposeServerlessAddressString(&name, &zone, address);
   }

   if ((err == noErr) && (name->dataLength == 0))
      err = kNullNameError;
   else if ((err == kSMPInvalidAddressString) && (name == nil))
      err = kNullNameError;

   if (err == noErr)
   {
      err = DoSetRStringProperty(callBlockPtr, prMyName, name, true);
      err = DoSetRStringProperty(callBlockPtr, prMyZone, zone, true);
      err = DoSetRStringProperty(callBlockPtr, prOldName, name, true);
   }
   else
   {
      tempErr = DoGetRStringPtrProperty(callBlockPtr, prOldAddress,
                                        &oldAddress);

      tempErr = DoSetRStringProperty(callBlockPtr, prMyAddress, oldAddress,
                                     true);
   }

   DisposeIfPtr(name);
   DisposeIfPtr(zone);
   DisposeIfPtr(oldAddress);
```

```
   DisposeIfPtr(address);

   return err;
}

/* ------------------------------------------------------------------------ */

static OSErr DoUpdateAddress(DETCallBlockPtr callBlockPtr)
{
   OSErr err = noErr;
   OSErr tempErr;
   RStringPtr name, zone, address;
   RStringPtr oldName, oldZone;
   short dirtyProperty;
   /* Initialize the variables. */
   name = nil;
   zone = nil;
   address = nil;
   oldName = nil;
   oldZone = nil;

   dirtyProperty = callBlockPtr->propertyDirtied.property;

   err = DoGetRStringPtrProperty(callBlockPtr, prMyName, &name);
   err = DoGetRStringPtrProperty(callBlockPtr, prMyZone, &zone);

   if (err == noErr)
   {
      if ((name->dataLength == 0) && (dirtyProperty == prMyName))
         err = kNullNameError;
   }

   if (err == noErr)
   {
      err = MyCreateAddressString(name, zone, &address);
   }

   if (err == noErr)
   {
      err = DoSetRStringProperty(callBlockPtr, prMyAddress, address, true);
      err = DoSetRStringProperty(callBlockPtr, prOldAddress, address, true);
      err = DoSetRStringProperty(callBlockPtr, prOldName, name, true);
   }
```

```
   else
   {
      /* Revert to the previous version. */

      tempErr = DoGetRStringPtrProperty(callBlockPtr, prOldName, &oldName);
      if (tempErr == noErr)
      {
      tempErr = DoSetRStringProperty(callBlockPtr, prMyName, oldName, true);
      }
   }

   DisposeIfPtr(oldName);
   DisposeIfPtr(oldZone);
   DisposeIfPtr(address);
   DisposeIfPtr(name);
   DisposeIfPtr(zone);

   return err;
}
```

Miscellaneous Subroutines

The subroutines in Listing 4-9 are called by one or more of the functions in the preceding sections.

Listing 4-9 Miscellaneous subroutines used by the address template code resource

```
/* Set value of property prInited. */
static void DoSetInited(DETCallBlockPtr callBlockPtr, Boolean inited)
{
   OSErr err = noErr;
   err = DoSetBooleanProperty(callBlockPtr, prInited, inited, false);
}

/* ------------------------------------------------------------------------ */
/* Get value of property prInited. */
static Boolean DoIsInited(DETCallBlockPtr callBlockPtr)
{
   OSErr err = noErr;
   Boolean inited = false;
   err = DoGetBooleanProperty(callBlockPtr, prInited, &inited);
```

```
   if (err != noErr)
      inited = false;

   return inited;
}
/* ------------------------------------------------------------------------ */
/* Error handler */
static void DoHandleError(DETCallBlockPtr callBlockPtr, OSErr err)
{
   if ((err == noErr) || (err == kDETDidNotHandle))
      return;
/* Call kDETcmdAboutToTalk callback routine. */
   AboutToTalk(callBlockPtr);

/* Display one of the error messages in the template. */
   switch (err)
   {
      case kNullNameError:
         DisplayErrorMessage(err, kSurfInfoPageAspect, 2);
         break;

      default:
         DisplayErrorMessage(err, kSurfInfoPageAspect, 1);
         break;
   }
}

/* ------------------------------------------------------------------------ */
/* Call kDETcmdSetPropertyRString callback routine. */
pascal OSErr DoSetRStringProperty(DETCallBlockPtr callBlockPtr,
                        short property,
                        RStringPtr newValue,
                        Boolean markAsChanged)
   {
   OSErr err;
   DETCallBackBlock cbb;

   cbb.setPropertyRString.reqFunction = kDETcmdSetPropertyRString;
   cbb.setPropertyRString.property = property;
   cbb.setPropertyRString.target.selector = kDETSelf;
   cbb.setPropertyRString.newValue = newValue;

   err = CallBackDET(callBlockPtr, &cbb);
```

```
   if ((err == noErr) && markAsChanged)
      {
      err = DoSetPropertyChanged(callBlockPtr, property, true);
      }

   return err;
   }

/* ---------------------------------------------------------------------- */
/* Call kDETcmdSetPropertyNumber callback routine. */
pascal OSErr DoSetNumProperty(DETCallBlockPtr callBlockPtr,
                              short property,
                              unsigned long newValue,
                              Boolean markAsChanged)
   {
   OSErr err;
   DETCallBackBlock cbb;

   cbb.setPropertyRString.reqFunction = kDETcmdSetPropertyNumber;
   cbb.setPropertyRString.property = property;
   cbb.setPropertyRString.target.selector = kDETSelf;
   cbb.setPropertyRString.newValue = newValue;

   err = CallBackDET(callBlockPtr, &cbb);

   if ((err == noErr) && markAsChanged)
      {
      err = DoSetPropertyChanged(callBlockPtr, property, true);
      }

   return err;
   }

/* ---------------------------------------------------------------------- */
/* Call kDETcmdSetPropertyNumber callback routine with a value of 0 or 1. */
pascal OSErr DoSetBooleanProperty(DETCallBlockPtr callBlockPtr,
                           short property,
                           Boolean value,
                           Boolean markChanged)
{
   OSErr err = noErr;

   err = DoSetNumProperty(callBlockPtr, property, value, markChanged);
```

```
   return err;
}

/* ------------------------------------------------------------------------ */
/* Call kDETcmdSetPropertyChanged callback routine. */
pascal OSErr DoSetPropertyChanged(DETCallBlockPtr callBlockPtr,
                        short property,
                        Boolean propertyChanged)
{
   OSErr err;
   DETCallBackBlock cbb;

   cbb.SetPropertyChanged.reqFunction = kDETcmdSetPropertyChanged;
   cbb.SetPropertyChanged.property = property;
   cbb.SetPropertyChanged.target.selector = kDETSelf;
   cbb.SetPropertyChanged.propertyChanged = propertyChanged;

   err = CallBackDET(callBlockPtr, &cbb);
   return err;
}

/* ------------------------------------------------------------------------ */
/* Call kDETcmdGetPropertyRString callback routine. */
pascal OSErr DoGetRStringProperty(DETCallBlockPtr callBlockPtr,
                                  short property,
                                  RString ***str)
{
   OSErr err;
   DETCallBackBlock cbb;

   cbb.getPropertyRString.reqFunction = kDETcmdGetPropertyRString;
   cbb.getPropertyRString.property = property;
   cbb.getPropertyRString.target.selector = kDETSelf;

   err = CallBackDET(callBlockPtr, &cbb);
   *str = cbb.getPropertyRString.propertyValue;

   return err;
```

```
/* ------------------------------------------------------------------------ */
/* Call kDETcmdGetPropertyNumber callback routine. */
pascal OSErr DoGetNumProperty(DETCallBlockPtr callBlockPtr,
                              short property,
                              long *value)
{
   OSErr err;
   DETCallBackBlock cbb;

   cbb.getPropertyNumber.reqFunction = kDETcmdGetPropertyNumber;
   cbb.getPropertyNumber.property = property;
   cbb.getPropertyNumber.target.selector = kDETSelf;

   err = CallBackDET(callBlockPtr, &cbb);

   *value = cbb.getPropertyNumber.propertyValue;

   return err;
}

/* ------------------------------------------------------------------------ */
/* Call kDETcmdGetPropertyNumber callback routine and return 0 or 1. */
pascal OSErr DoGetBooleanProperty(DETCallBlockPtr callBlockPtr,
                          short property,
                          Boolean* value)
{
   OSErr err = noErr;
   long number;

   err = DoGetNumProperty(callBlockPtr, property, &number);
   *value = (number == 1);

   return err;
}

/* ------------------------------------------------------------------------ */
/* Call kDETcmdGetPropertyRString callback routine and convert handle to
    pointer. */
pascal OSErr DoGetRStringPtrProperty(DETCallBlockPtr callBlockPtr,
                                     short property,
                                     RStringPtr* str)
{
   OSErr err = noErr;
```

```
   RStringHandle stringH = nil;
   RStringPtr stringP = nil;

   err = DoGetRStringProperty(callBlockPtr, property, &stringH);

   if (err == noErr)
   {
      err = DoRStringHandleToPtr(stringH, &stringP);
   }

   DisposeHandle((Handle) stringH);
   *str = stringP;

   return err;
}

/* ---------------------------------------------------------------------- */
/* Convert an RString handle to a pointer. */
pascal OSErr DoRStringHandleToPtr(RStringHandle stringH,
                        RStringPtr* string)
{
   OSErr err = noErr;
   RStringPtr stringP = nil;

   stringP = (RStringPtr) NewPtr(sizeof(ProtoRString) +
                                 (*stringH)->dataLength);

   HLock((Handle) stringH);
   err = OCECopyRString(*stringH, stringP, (*stringH)->dataLength);
   HUnlock((Handle) stringH);

   *string = stringP;
   return err;
}
```

SAM Setup Reference

This section lists and describes the contents of the PowerTalk Setup catalog and the attributes in records of the following types:

- Setup
- MSAM
- CSAM
- Mail Service
- Catalog
- Combined

Following the descriptions of these records and attributes, this section describes all of the properties and resources that you must include in your setup template.

The PowerTalk Setup Catalog

The information in the PowerTalk Setup catalog completely describes the AOCE services available on the Macintosh computer on which the Setup catalog is located. The records in the Setup catalog contain information about the installed CSAMs, the catalogs associated with those CSAMs, the installed personal MSAMs, and the messaging services associated with those MSAMs. Each CSAM and personal MSAM is represented by a record, and each personal MSAM mail slot and CSAM catalog is represented by a record (a single Combined record can represent both a mail slot and a catalog). In addition, there is a special Setup record that ties everything together.

When a user installs the PowerTalk software on his or her Macintosh, PowerTalk software creates the Setup catalog. The Setup catalog initially contains a single record called the ***Setup record.*** As the user adds catalog or messaging services, additional records are needed in the Setup catalog to specify these services.

Many of the records in the Setup catalog refer to other records in the Setup catalog by means of a ***record reference.*** A record reference is an attribute whose value consists of a packed record ID, of which only the creation ID is used to identify a given record. Except for parent MSAM record attributes, which your setup template creates, the PowerTalk Key Chain manages record references; you shouldn't need to examine or manipulate them.

Setup templates create or modify many of the records in the Setup catalog. The sections that follow describe the types of records in the Setup catalog and the contents of those records. The sections “Adding Catalog and Mail Services” beginning on page 4-5 and “Modifying an Existing Service” on page 4-30 explain how you actually create and modify the records.

Table 4-1 lists the standard types of records contained in the Setup catalog, provides the corresponding record type index where applicable, and notes who creates a record of that type. Record type indexes are described in the chapter "AOCE Utilities" in the book *Inside Macintosh: AOCE Application Interfaces*.

Table 4-1 Setup-catalog record types

Type of record	Record type index	Created by
Setup	`kSetupRecTypeNum`	AOCE
MSAM	`kMSAMRecTypeNum`	Setup template (by calling the `DirAddRecord` function if this record does not already exist)
CSAM	`kDSAMRecTypeNum`	Setup template (by calling the `DirAddDSAM` function)
Mail Service (also known as a *slot record*)	N/A	Key Chain; main aspect template provided by MSAM-only setup templates
Catalog	N/A	Key Chain; main aspect template provided by MSAM-only and CSAM-only setup templates
Combined	N/A	Key Chain; main aspect template provided by combined MSAM and CSAM setup templates

The Setup Record

There is a single Setup record in the Setup catalog. It contains record references to all of the records in the PowerTalk Setup catalog that represent slots, catalogs, and other items that show up in the PowerTalk Key Chain. It is identified by the record type index constant `kSetupRecTypeNum`. The Key Chain sets up and maintains the Setup record; you do not manipulate it or read it.

The MSAM Record

An MSAM record represents a personal MSAM. The setup template for a given MSAM creates this record. The record name is the same as the name of the file containing the personal MSAM at the time you create the record. (Once the MSAM has been created and configured, the user can change the filename without affecting the MSAM record). An MSAM record is identified by the record type index constant `kMSAMRecTypeNum`. Table 4-2 shows the attributes for an MSAM record.

Table 4-2 Attributes of an MSAM record

Attribute type and index	Data type	Description	Written by
AOCE version `kVersionAttrTypeNum`			
	`long`	AOCE version number.	Setup template
Gateway file ID `kGatewayFileIDAttrTypeNum`			
	`long`	The file ID of the personal MSAM file. To activate a mail slot, you need to find the slot's Mail Service record. To do so, you compare your file's file ID with the file ID stored in the gateway file ID attribute in each MSAM record. (If no Mail Service record exists for this file, you must create one.)	Setup template
Mail service `kMailServiceAttrTypeNum`			
	`PackedRecordID`	A record reference to a Mail Service record that represents a slot belonging to this personal MSAM. An MSAM record contains one mail service attribute for each slot associated with the MSAM.	Setup template

An MSAM record may also contain MSAM-specific information in attributes added by the personal MSAM, its setup template, or both.

The CSAM Record

A CSAM record represents a CSAM. A setup template creates this record by calling the `DirAddDSAM` function. The record name is the same as the name of the file that contains the CSAM. A CSAM record is identified by the record type index constant `kDSAMRecTypeNum`. Table 4-3 shows the attributes for a CSAM record.

Table 4-3 Attributes of a CSAM record

Attribute type and index	Data type	Description	Written by
CSAM alias `kDSAMFileAliasAttrTypeNum`			
	Private to AOCE	An alias to the CSAM file, created and used by AOCE software.	Setup template calls `DirAddDSAM` function
Catalog `kDirectoryAttrTypeNum`			
	`PackedRecordID`	A record reference to a Catalog record that represents a catalog available through this CSAM. A CSAM record contains one catalog attribute for each catalog associated with the CSAM.	Setup template calls `DirAddDSAMDirectory` function

A CSAM record may also contain CSAM-specific information added by the CSAM, its setup template, or both.

The Mail Service Record

A Mail Service record (also known as a *slot record*) contains information about a mail slot. The Key Chain creates this record, using a main aspect template provided by your setup template, when your setup template adds a mail service only. When your setup template adds a combined mail and catalog service, the mail slot information is contained in a Combined record rather than a Mail Service record.

Your main aspect template for the Mail Service record must specify the record type "aoce Mail Service*xxxx*" where *xxxx* is the address extension type of the messaging system to which your MSAM provides access. You can use the constant `kMailServiceRecTypeBody` to do so; for example, the following fragment of an aspect template creates a record of type "aoce Mail ServiceWAVE":

```
#define kServiceRecordType kMailServiceRecTypeBody "WAVE"

resource 'deta' (kSurfWriterAspect, purgeable)
{
   0, dropCheckConflicts, isMainAspect
};

resource 'rstr' (kSurfWriterAspect + kDETRecordType, purgeable)
{
   kServiceRecordType
};
```

Table 4-4 shows the attributes for a Mail Service record.

Table 4-4 Attributes of a Mail Service record

Attribute type and index	Data type	Description	Written by
AOCE version `kVersionAttrTypeNum`			
	`long`	AOCE version number.	Key Chain
Associated catalog `kAssoDirectoryAttrTypeNum`			
	`PackedRecordID`	A record reference to the record that represents the catalog with which this slot is associated. AOCE needs the Catalog record to route messages; the Setup catalog must contain the Catalog record before your MSAM can be properly set up.	Key Chain
Parent MSAM `kParentMSAMAttrTypeNum`			
	`PackedRecordID`	A record reference to the record that represents the MSAM to which this slot belongs.	Setup template
Slot ID `kSlotIDAttrTypeNum`			
	`SlotID`	The slot ID for this mail slot.	MSAM
Standard slot information `kStdSlotInfoAttrTypeNum`			
	`MailStandardSlotInfoAttribute`	A structure that contains information about the slot, such as when to log on to the external messaging system.	Setup template

The Mail Service record may also contain other slot-specific information added by the personal MSAM, its setup template, or both.

The Catalog Record

A Catalog record represents a catalog to which the user has access. The Key Chain creates this record, using a main aspect template provided by your setup template, when your setup template adds a mail service only or a catalog service only. When your setup template adds a combined mail and catalog service, the catalog information is contained in a Combined record rather than a Mail Service record, as described in the next section.

Your main aspect template for the Catalog record must specify the record type "aoce Directory*xxxx*". If the catalog is associated with a mail slot, *xxxx* is the address extension type of the external messaging system to which the slot's MSAM provides access. If the catalog is not associated with a slot, *xxxx* is the `signature` field of the catalog discriminator (`DirDiscriminator` structure).

You can use the constant `kDirectoryRecTypeBody` to assign the record type; for example, the following fragment of an aspect template creates a record of type "aoce DirectoryWAVE":

```
#define kServiceRecordType kDirectoryRecTypeBody "WAVE"

resource 'deta' (kSurfWriterAspect, purgeable)
{
   0, dropCheckConflicts, isMainAspect
};

resource 'rstr' (kSurfWriterAspect + kDETRecordType, purgeable)
{
   kServiceRecordType
};
```

Table 4-5 shows the attributes for a Catalog record. Note that some attributes are used in Catalog records only for stand-alone MSAMs, some are used in Catalog records only for stand-alone CSAMs, and some are used both for MSAMs and CSAMs. When you install a combined MSAM and CSAM, you provide a template for a Combined record rather than for a Catalog record (see Table 4-6 on page 4-70).

Table 4-5 Attributes of a Catalog record

Attribute type and index	Data type	Description	Written by	Used for
Version `kVersionAttrTypeNum`				
	`long`	Version number of the Key Chain at the time the record was created	Key Chain	CSAM
Associated mail service `kAssoMailServiceAttrTypeNum`				
	`PackedRecordID`	Record reference to the Mail Service record associated with this catalog	Key Chain	MSAM
Parent CSAM `kParentDSAMAttrTypeNum`				
	`PackedRecordID`	Record reference to CSAM record for this catalog	Setup template calls `DirAddDSAMDirectory` function	CSAM

continued

Table 4-5 Attributes of a Catalog record (continued)

Attribute type and index	Data type	Description	Written by	Used for
Discriminator `kDiscriminatorAttrTypeNum`				
	`DirDiscriminator`	This catalog's discriminator value	Setup template (CSAM template calls `DirAddDSAMDirectory` function)	MSAM CSAM
Capability flags `kSFlagsAttrTypeNum`				
	`long`	This catalog's capability flags	Setup template calls `DirAddDSAMDirectory` function	CSAM
Comment `kCommentAttrTypeNum`				
	`RString`	Comment for your use	Setup template	MSAM CSAM
Real name `kRealNameAttrTypeNum`				
	`RString`	Name of this catalog for your use	Setup template	MSAM CSAM
User's record ID `kDirUserRIDAttrTypeNum`				
	`RecordID`	User's record ID	Setup template calls `OCESetupAddDirectoryInfo` function	MSAM CSAM
Native name `kDirNativeNameAttrTypeNum`				
	`RString`	User's name or account name in the external catalog; for your use	Setup template	MSAM CSAM
User's key `kDirUserKeyAttrTypeNum`				
	Private to AOCE	User's encrypted password	Setup template calls `OCESetupAddDirectoryInfo` function	MSAM

continued

Table 4-5 Attributes of a Catalog record (continued)

Attribute type and index	Data type	Description	Written by	Used for
Private data `kPrivateDataAttrTypeNum`				
	Binary data of any length (to maximum size of attribute)	Data for your use; for example, information about address formats	Setup template	MSAM CSAM

The Combined Record

A Combined record represents a mail slot and a catalog in a single record. The Key Chain creates this record, using a main aspect template provided by your setup template, when your setup template adds a combined MSAM and CSAM.

Your main aspect template for the Combined record must specify the record type "aoce Combined*xxxx*" where *xxxx* is the address extension type of the external messaging system to which the slot's MSAM provides access.

You can use the constant `kCombinedRecTypeBody` to assign the record type; for example, the following fragment of an aspect template creates a record of type "aoce CombinedWAVE":

```
#define kServiceRecordType kCombinedRecTypeBody "WAVE"

resource 'deta' (kSurfWriterAspect, purgeable)
{
   0, dropCheckConflicts, isMainAspect
};

resource 'rstr' (kSurfWriterAspect + kDETRecordType, purgeable)
{
   kServiceRecordType
};
```

Table 4-6 shows the attributes for a Combined record.

Table 4-6 Attributes of a Combined record

Attribute type and index	Data type	Description	Written by
Version `kVersionAttrTypeNum`			
	`long`	Version number of the Key Chain at the time the record was created.	Key Chain

continued

Table 4-6 Attributes of a Combined record (continued)

Attribute type and index	Data type	Description	Written by
Associated catalog `kAssoDirectoryAttrTypeNum`			
	`PackedRecordID`	A record reference to the record that represents the catalog with which this slot is associated. For Combined records, this attribute points back to the Combined record itself.	Key Chain
Associated mail service `kAssoMailServiceAttrTypeNum`			
	`PackedRecordID`	Record reference to the Mail Service record associated with this catalog. For Combined records, this attribute points back to the Combined record itself.	Key Chain
Parent CSAM `kParentDSAMAttrTypeNum`			
	`PackedRecordID`	Record reference to the CSAM record for this catalog.	Setup template calls `DirAddDSAMDirectory` function
Parent MSAM `kParentMSAMAttrTypeNum`			
	`PackedRecordID`	A record reference to the record that represents the MSAM to which this slot belongs.	Setup template
Slot ID `kSlotIDAttrTypeNum`			
	`SlotID`	The slot ID for this mail slot.	MSAM
Standard slot information `kStdSlotInfoAttrTypeNum`			
	`MailStandardSlotInfoAttribute`	A structure that contains information about the slot, such as when to log on to the external messaging system.	Setup template

continued

Table 4-6 Attributes of a Combined record (continued)

Attribute type and index	Data type	Description	Written by
Discriminator `kDiscriminatorAttrTypeNum`	`DirDiscriminator`	This catalog's discriminator value.	Setup template calls `DirAddDSAMDirectory` function
Capability flags `kSFlagsAttrTypeNum`	`long`	This catalog's capability flags.	Setup template calls `DirAddDSAMDirectory` function
Comment `kCommentAttrTypeNum`	`RString`	Displayable comment about this catalog; for example, the time the catalog was installed.	Setup template
Real name `kRealNameAttrTypeNum`	`RString`	Name of this catalog for your use.	Setup template
User's record ID `kDirUserRIDAttrTypeNum`	`RecordID`	User's record ID.	Setup template calls `OCESetupAddDirectoryInfo` function
Native name `kDirNativeNameAttrTypeNum`	`RString`	User's name or account name in the external catalog; for your use.	Setup template
User's key `kDirUserKeyAttrTypeNum`	Private to AOCE	User's encrypted password.	Setup template calls `OCESetupAddDirectoryInfo` function
Private data `kPrivateDataAttrTypeNum`	Binary data of any length (to maximum size of attribute)	Data for your use; for example, information about address formats.	Setup template

The Setup Template Resources

Every CSAM and personal MSAM must include a setup template in the resource fork of the SAM file. The setup template provides the human interface that allows a user to add or remove the SAM and the services that it supports. In response to user input, the template creates and modifies records in the Setup catalog.

A setup template consists of an aspect template and at least one information page template. To learn how to write an AOCE template, read the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces*. This section covers only those topics that are specific to setup templates.

Table 4-7 shows the resources required for the setup aspect template. Only those resources that are unique to setup templates or that must have specific values in setup templates are described here. For descriptions of the other required resources and for a complete list of all the resources you can use in aspect templates, see the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces*.

Table 4-7 Required resources for setup aspect templates

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
`'deta'`	`0`	Identifies template as main aspect and provides a base resource ID.
`'rstr'`	`kDETTemplateName`	Name of template.
`'rstr'`	`kDETRecordType`	Type of record to which the template applies.
`'rstr'`	`kDETAspectName`	The name displayed in the Key Chain in the Service field.
`'rstr'`	`kSAMAspectKind`	The kind of service as shown in the Kind field of the Key Chain.
`'detn'`	`kSAMAspectCannotDelete`	A property that detemines whether the user can delete the slot or catalog set up by this aspect.
`'rstr'`	`kSAMAspectUserName`	The string the Key Chain displays in the Name field.
`'sami'`	`kSAMAspectSlotCreationInfo`	The information needed by the Key Chain to create and delete slots and catalogs.
Icon suite	`kDETAspectMainBitmap`	Suite of icons.
`'rstr'`	`kDETAspectKind`	The kind of record as shown in the Get Info dialog box. Neither the code resource nor the user can change this value.
`'rstr'`	`kDETAspectWhatIs`	Help-balloon string for objects of the type described by this aspect when they appear in the Key Chain.
`'detc'`	`kDETAspectCode`	A code resource.

IMPORTANT
Because of these additional required resources, your own properties should start at offset `kSAMFirstDevProperty` rather than at `kFirstDevProperty`. ▲

The rest of this section describes the resources required for setup aspect templates. For a complete description of the resources you can use in any AOCE aspect template, see the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces*.

Aspect Signature Resource

You must supply a main aspect template for Mail Service records, Catalog records, and Combined records required for your setup template (see Table 4-1 on page 4-64). The aspect signature resource provides the base resource ID for the aspect template and specifies that the template is a main aspect template. Because users cannot drag records out of the Key Chain or drop them in, the drop-related fields of the aspect signature resource are not significant. The signature resource for an aspect template is of type `'deta'`.

```
resource 'deta' (kSurfWriterAspect, purgeable)
{
   0, dropCheckConflicts, isMainAspect
};
```

kDETTemplateName

The template name resource is required of all templates. It has a resource ID with an offset of `kDETTemplateName` from the template's base (signature) resource ID.

```
resource 'rstr' (kSurfWriterAspect+kDETTemplateName, purgeable) {
   "kAspectName"
};
```

Your information page templates must use this name to refer to the aspect template that provides their property values.

kDETRecordType

The record-type resource specifies the record type to which the aspect template applies. The record-type resource has a resource ID with an offset of `kDETRecordType` from the template's base resource ID.

```
#define kServiceRecordType kMailServiceRecTypeBody "WAVE"

resource 'rstr' (kSurfWriterAspect + kDETRecordType, purgeable)
{
   kServiceRecordType
};
```

A main aspect template for a Mail Service record must specify the record type "aoce Mail Service*xxxx*" where *xxxx* is the address extension type of the messaging system to which your MSAM provides access. A main aspect template for a Catalog record must specify the record type "aoce Directory*xxxx*". If the catalog is associated with a mail slot, *xxxx* is the extension type of the external messaging system to which the slot's MSAM provides access. If the catalog is not associated with a slot, *xxxx* is the `signature` field of the catalog discriminator (`DirDiscriminator` structure). A main aspect template for a Combined record must specify the record type "aoce Combined*xxxx*" where *xxxx* is the extension type of the external messaging system to which the slot's MSAM provides access. You can use the following constants when assigning a record type:

```
#define  kDirectoryRecTypeBody      "aoce Directory"
#define  kMailServiceRecTypeBody    "aoce Mail Service"
#define  kCombinedRecTypeBody       "aoce Combined"
```

kDETAspectName

A setup aspect template should specify a default name for the CE to display in the Service field in the Key Chain. To provide this name, use an `RString` resource with an offset of `kDETAspectName` from the template's base resource ID.

Note
For main aspect templates for records other than setup templates, the CE automatically sets the `kDETAspectName` property to be the name of the record. However, for records in the Key Chain, your template must provide a resource to set this property explicitly. ◆

```
resource 'rstr' (kSurfWriterAspect+kDETAspectName, purgeable)
{
   "New Mail Server"
};
```

Your setup information page template can allow the user to change this name to the name of the mail server or catalog server on which he or she has an account.

kDETAspectKind

Specify the kind of the record as it is to be displayed in a Get Info dialog box with an `RString` resource with an offset of `kDETAspectKind` from the template's base resource ID.

```
resource 'rstr' (kSurfWriterAspect+kDETAspectKind, purgeable)
{
   "SurfWriter Mail Service"
};
```

This resource is the only source for this information. Neither your code resource nor the user can change the value you specify in this resource. Unlike AOCE dNode windows, however, the Key Chain does not use the `kDETAspectKind` resource to determine what to display in the Kind field. The Key Chain uses the `kSAMAspectKind` resource (described next) for that purpose.

kSAMAspectKind

A setup aspect template must specify the name the CE should display in the Kind field in the Key Chain. To provide this name, use an `RString` resource with an offset of `kSAMAspectKind` from the template's base resource ID.

```
resource 'rstr' (kSurfWriterAspect+kSAMAspectKind, purgeable)
{
   "SurfWriter Mail Service"
};
```

This resource is the only source for this information. Your code resource can change the value you specify in this resource. You must also include a `kDETAspectKind` resource to specify the kind of the record as it is to be displayed in a Get Info dialog box.

kSAMAspectUserName

A setup aspect template should specify the default name the CE should display in the Name field in the Key Chain. To provide this name, use an `RString` resource with an offset of `kSAMAspectUserName` from the template's base resource ID.

```
resource 'rstr' (kSurfWriterAspect+kSAMAspectUserName, purgeable)
{
   "SurfWriter User"
};
```

Your setup information page template can allow the user to change this name to the account name on the mail system or catalog server. If your system does not use account names, you should use the name of the owner of the Key Chain for this property. You can obtain this name from the Setup record, where it is stored in an attribute of type "Local Name". The attribute type index for this attribute is `kLocalNameAttrTypeNum`.

kSAMAspectCannotDelete

The `kSAMAspectCannotDelete` property indicates whether the slot or catalog associated with this aspect can be deleted. A property value of 0 indicates that the slot or catalog can be deleted. Otherwise, it cannot be deleted. The default value of this property is 0. Your setup template can set the value of this property to prevent the user from deleting the slot or catalog once it has been added.

```
resource 'detn' (kSurfWriterAspect+kSAMAspectCannotDelete,
purgeable)
{
   1
};
```

kSAMAspectSlotCreationInfo

The slot creation information resource gives the Key Chain the information it needs to create and delete MSAM slots and CSAM catalogs. To provide this information, use a resource of type `'sami'` with an offset of `kSAMAspectSlotCreationInfo` from the template's base resource ID. This resource has the following Rez type definition:

```
type 'sami' {
   integer;                  // max number of catalogs/slots
   longint;                  // catalog signature, MSAM type
   byte notMSAM, servesMSAM;  // an MSAM template?
   byte notDSAM, servesDSAM;  // a CSAM template?
   rstring;                  // display when user clicks Add
   align word;
   rstring;                  // new record name
   align word;
};
```

The `integer` value is the maximum number of slots, catalogs, or combined services that your SAM can support. Set this to 0 if you can support an unlimited number of slots or catalogs.

The `longint` value identifies your type of service. For catalogs, this is the value of the `signature` field of the catalog discriminator (`DirDiscriminator` structure; see the chapter "AOCE Utilities" in *Inside Macintosh: AOCE Application Interfaces*). For slots, this is the extension type of the addresses. Addresses and address extension types are described in the chapter "Messaging Service Access Modules" in this book. For a catalog and mail service to work together, the catalog discriminator and address extension type values must be the same.

The two `byte` values specify whether your SAM is an MSAM or a CSAM. If it is a combined MSAM and CSAM, specify both `servesMSAM` and `servesDSAM` for these values.

The first `RString` value specifies the text for the dialog box that the Key Chain displays when the user clicks the Add button. This value in the `kSAMAspectSlotCreationInfo` resource replaces the `kDETAspectNewMenuName` resource used in other (non-setup) main aspect templates.

The second `RString` value specifies the name the Key Chain assigns initially to new records of this type. (The user can use the Key Chain information page to rename this record.) This value in the `kSAMAspectSlotCreationInfo` resource replaces the `kDETAspectNewEntryName` resource used in other (non-setup) main aspect templates.

Here is an example of a `kSAMAspectSlotCreationInfo` resource:

```
#define kSignature   'WAVE'
resource 'sami' (kSurfWriterAspect + kSAMAspectSlotCreationInfo,
                 purgeable)
{
   2,
   kSignature,
   servesMSAM,
   servesDSAM,
   "SurfWriter Combined Service",
   "untitled combined SurfWriter"
};
```

kDETAspectMainBitmap

Every main aspect template, including a setup template, must include an icon suite with a resource ID that has an offset of `kDETAspectMainBitmap` from the template's base resource ID. Suppose, for example, that you prepared an icon suite in a ResEdit file named SurfWriterIcons, that all of your icon resources had resource IDs of 0, and that your resource base ID was `kSurfWriterAspect`. In this case, you could use the following code to include the icon suite in your setup aspect template:

```
include "SurfWriterIcons" 'ICN#'(0) as
                  'ICN#'(kMainAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'icl4'(0) as
                  'icl4'(kMainAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'icl8'(0) as
                  'icl8'(kMainAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics#'(0) as
                  'ics#'(kMainAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics4'(0) as
                  'ics4'(kMainAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'ics8'(0) as
                  'ics8'(kMainAspect+kDETAspectMainBitmap, purgeable);
include "SurfWriterIcons" 'SICN'(0) as
                  'SICN'(kMainAspect+kDETAspectMainBitmap, purgeable);
```

The icon suite must be included in the main aspect template and cannot be changed from a code resource or by the user.

kDETAspectWhatIs

Each setup main-aspect template must provide a help-balloon string. The Key Chain displays this string when the user enables Balloon Help online assistance and moves the cursor over an entry in the Key Chain to which this main aspect applies. The help-balloon string is in an `RString` resource with an offset `kDETAspectWhatIs` from the template's base resource ID.

```
resource 'rstr' (kSurfWriterAspect+kDETAspectWhatIs, purgeable) {
   "Contains information about this key, which represents a
     SurfWriter combined mail and catalog service."
};
```

kDETAspectCode

Every setup aspect template must include a code resource with an offset of `kDETAspectCode` from the template's base resource ID. For example, to include code that has been compiled and saved as the resource `SurfWriterCode` of type `'detc'` with a resource ID of 0, add the following line to the setup aspect template:

```
include "SurfWriterCode" 'detc'(0) as
    'detc'(kSurfWriterAspect+kDETAspectCode, purgeable);
```

Your setup template code resource must call a variety of Collaboration toolbox functions and AOCE template callback functions to create records and attributes. See "Adding Catalog and Mail Services" beginning on page 4-5 for a description of each step involved in setting up catalog and mail services. Code resources and template callback functions are described in the chapter "AOCE Templates" in *Inside Macintosh: AOCE Application Interfaces*.

The Address Template

Every MSAM must include an address template in the resource fork of the MSAM file. The template provides the human interface that allows a user to view, create, and edit the addresses the MSAM needs to send letters to recipients on its messaging system.

An address template consists of an aspect template and at least one information page template.

The lookup table (`'dett'` pattern) for an address must end with a pattern element of type `'Pref'`. This custom element type lets the Electronic Addresses information page set the preferred address radio buttons correctly.

The standard address information page is 259 pixels wide and 200 pixels high. It has a page-selection pop-up menu at location (8, 56, 30, 206) (top, left, bottom, right). It has a page-identifying large icon at (8, 8, 40, 40). Within the page are two radio buttons labeled "View as:", a Fields radio button and a String radio button. The string "View as:" is at location (49, 56, 63, 106). The Fields radio button is at location (48, 111, 64, 154). The String radio button is at location (48, 164, 64, 209). Between the view-as selector and the data is a dotted line, at location (72, 8, 73, 251).

Addresses with all types of tags are forwarded to the drop-send aspect by a built-in forwarder. For this reason, your address template does not need to handle drops.

For an example of an address template, see "Writing and Modifying Addresses" beginning on page 4-30.