Technical Note TN1002
On Launching an App with a Document

CONTENTS

This Technote describes how to launch another application from inside your app and then open a document using that app. Preferably, developers would like to utilize the signature information from the document to find the application.

Developers writing applications that sub-launch other applications will find the material contained herein lets them easily extend the functionality of their applications to include the ability to launch applications with the specific documents they want to be opened.

This Note combines information found in the chapter "Creating and Sending Apple Events" in *Inside Macintosh: Interapplication Communications* and the chapter "Desktop Manager" in *Inside Macintosh: More Macintosh Toolbox* . The format of the Open Documents event is described in the section "Handling the Required Apple Events" (4-11) in *Inside Macintosh: Interapplication Communication* .

[Jul 24 2000]

About Launching an Application with a Document

To launch an application with a document, the following steps are required:

1. Find the appropriate application for opening the document
2. Try sending the open documents event to the running application, and

3. If Step 2 fails, launch the application with the appropriate parameters.

Finding the Right Application for the Document

Every document has type and creator bytes associated with it. The first step, therefore, in finding what application to use is to take a look at the creator bytes. You can find this value by using the `FSpGetFInfo` call:

```
err = FSpGetFInfo(&spec, &fndrInfo);
```

`fndrInfo.fdCreator` contains the creator bytes for the document.

When specifying the destination address for the open document Apple Event, the creator bytes are used to create a descriptor containing a `typeApplSignature` record. If the `AESend` indicates that no such application with that creator is running, then the creator bytes can be used to search the desktop files on attached volumes for the correct application to launch.

Using a Preferred Application to Open the Document

Sometimes, you may want to open a document using an application other than the one referred to by its creator bytes. For example, you may want to open every text document you encounter using a program such as AppleWorks. In that case, you need to provide some internal means of mapping file types to creator types inside of your application that overrides the creator bytes returned by `FSpGetFInfo`. The code snippet in this Technote provides a means for doing this by allowing you to define a mapping function for re-mapping the creator type used to open a document.

Here is how you would define a simple mapping function that will work with `LaunchTheDocument`:

```
void MyMapCreator(FSSpec *document, OSType type,
     OSType *creator) {
             /* open all text files with
             AppleWorks, creator = 'BOBO' */
     if (type == 'TEXT')
         *creator = 'BOBO';
 }
```

Mapping functions receive three parameters: a `FSSpec` record referring to the document itself -- in case you want to look at the document's data or name to establish its type -- and the document's type and creator bytes. Mapping functions provide a simple way for your application to change the creator bytes that will be used to find the appropriate application before the search for the application begins. In this example, all documents of type `'TEXT'` are re-mapped to the creator bytes for AppleWorks, telling `LaunchTheDocument` to ask AppleWorks to open all text documents. The collected fragments in a code snippet provided later in this Technote give you an example of how to define a creator mapping function.

Sending the Open Documents Event

As described in *Inside Macintosh: Interapplication Communication* (4-13), the open documents event contains one list of alias records referring to documents to be opened by the target application. The following description takes you through the step-by-step construction of such an event.

First, create an address descriptor record (`appAddr`), using the creator bytes (`fndrInfo.fdCreator`):

```
        err = AECreateDesc(typeApplSignature,
             (Ptr) &fndrInfo.fdCreator,
             sizeof(OSType), &appAddr);
```

Then you use this address record to specify the target application when you create the Apple Event (`theEvent`):

```
        err = AECreateAppleEvent(kCoreEventClass,
             kAEOpenDocuments, &appAddr,
             kAutoGenerateReturnID, kAnyTransactionID,
             &theEvent);
```

Because you are opening a single document, you need to create a list containing an alias record referring to the file you want to open. Even though you're only asking the target application to open one document, you still create a list containing only one alias record to conform to the definition of an open documents Apple Event.

First, you create a list (`fileList`):

```
        err = AECreateList(NULL, 0, false, &fileList);
```

and a descriptor record (`fileListElt`) containing an alias to our document (`the_file`):

```
        err = NewAlias(NULL, &the_file, &docAlias);

         HLock((Handle) docAlias);
         err = AECreateDesc(typeAlias, (Ptr) (*docAlias),
             GetHandleSize((Handle) docAlias), &fileListElt);
         HUnlock((Handle) docAlias);
```

Then, you add the descriptor record (`fileListElt`) containing the alias to the list (`fileList`):

```
        err = AEPutDesc(&fileList, 0, &fileListElt);
```

Now that you have the Apple Event (`theEvent`), and the list (`fileList`), containing an alias record referring to the file you want opened, you can take the list and put it into the Apple Event (`theEvent`) as an Apple Event parameter:

```
        err = AEPutParamDesc(&theEvent,
             keyDirectObject, &fileList);
```

At this point the open documents event (`theEvent`) has been set up correctly and you can send it to the application:

```
        err = AESend(&theEvent, &eventReply,
             kAENoReply, kAENormalPriority, kNoTimeOut,
             NULL, NULL);
```

When the application is not running

When `AESend` returns an error result of `connectionInvalid`, this means the application with the specified creator bytes is not currently registered with the Process Manager. In this case, the correct application must be launched before it can be sent the open document event. The first thing to do here is search through the desktop files on connected volumes using the `PBDTGetAPPL` routine to find the application with the correct creator bytes for the document.

The following code fragment illustrates how this can be done.

```
       /* IsRemoteVolume can be used to find out if the
        volume referred to by vRefNum is a remote volume
        located somewhere on a network. the volume's attribute
        flags (copied from the GetVolParmsInfoBuffer structure)
        are returned in the longword pointed to by vMAttrib. */
    OSErr IsRemoteVolume(short vRefNum, Boolean *isRemote, long *vMAttrib) {
        HParamBlockRec volPB;
        GetVolParmsInfoBuffer volinfo;
        OSErr err;
        volPB.ioParam.ioVRefNum = vRefNum;
        volPB.ioParam.ioNamePtr = NULL;
        volPB.ioParam.ioBuffer = (Ptr) &volinfo;
        volPB.ioParam.ioReqCount = sizeof(volinfo);
        err = PBHGetVolParmsSync(&volPB);
        if (err == noErr) {
            *isRemote = (volinfo.vMServerAdr != 0);
            *vMAttrib = volinfo.vMAttrib;
        }
        return err;
    }


        /* BuildVolumeList fills the array pointed to by vols with
        a list of the currently mounted volumes.  If includeRemote
        is true, then remote server volumes will be included in
        the list.  When remote server volumes are included in the
        list, they will be added to the end of the list.  On entry,
        *count should contain the size of the array pointed to by
        vols.  On exit, *count will be set to the number of id numbers
        placed in the array. If vMAttribMask is non-zero, then
        only volumes with matching attributes are added to the
        list of volumes. bits in the vMAttribMask should use the
        same encoding as bits in the vMAttrib field of
        the GetVolParmsInfoBuffer structure. */
    OSErr BuildVolumeList(Boolean includeRemote, short *vols,
            long *count, long vMAttribMask) {
        HParamBlockRec volPB;
        Boolean isRemote;
        OSErr err;
        long nlocal, nremote;

            /* set up and check parameters */
        volPB.volumeParam.ioNamePtr = NULL;
        nlocal = nremote = 0;
        if (*count == 0) return noErr;

            /* iterate through volumes */
        for (volPB.volumeParam.ioVolIndex = 1;
            PBHGetVInfoSync(&volPB) == noErr;
            volPB.volumeParam.ioVolIndex++) {

                    /* skip remote volumes, if necessary */
            err = IsRemoteVolume(volPB.volumeParam.ioVRefNum, &isRemote);
            if (err != noErr) goto bail;
            if ( ( includeRemote || ! isRemote )
            && (vMAttrib & vMAttribMask) == vMAttribMask ) {

                    /* add local volumes at the front, remote
                    volumes at the end */
                if (isRemote)
                    vols[nlocal + nremote++] = volPB.volumeParam.ioVRefNum;
                else {
                    if (nremote > 0)
                        BlockMoveData(vols+nlocal, vols+nlocal+1,
                            nremote*sizeof(short));
                    vols[nlocal++] = volPB.volumeParam.ioVRefNum;
                }

                    /* list full? */
                if ((nlocal + nremote) >= *count) break;
            }
        }
    bail:
        *count = (nlocal + nremote);
        return err;
    }


        /* FindApplication iterates through mounted volumes
        searching for an application with the given creator
        type.  If includeRemote is true, then remote volumes
        will be searched (after local ones) for an application
        with the creator type. */

    #define kMaxVols 20

    OSErr FindApplication(OSType appCreator, Boolean includeRemote,
            FSSpec *appSpec) {
        short rRefNums[kMaxVols];
        long i, volCount;
        DTPBRec desktopPB;
        Str255 appName;
        OSErr err;

            /* get a list of volumes - with desktop files */
        volCount = kMaxVols;
        err = BuildVolumeList(includeRemote, rRefNums, &volCount,
            (1<<bHasDesktopMgr) );
        if (err != noErr) return err;
```

```
            /* iterate through the list */
        for (i=0; i<volCount; i++) {

                /* has a desktop file? */
            desktopPB.ioCompletion = NULL;
            desktopPB.ioVRefNum = rRefNums[i];
            desktopPB.ioNamePtr = NULL;
            desktopPB.ioIndex = 0;
            err = PBDTGetPath(&desktopPB);
            if (err != noErr) continue;

                /* has the correct app?? */
            desktopPB.ioFileCreator = appCreator;
            desktopPB.ioNamePtr = appName;
            err = PBDTGetAPPLSync(&desktopPB);
            if (err != noErr) continue;

                /* make a file spec referring to it */
            err = FSMakeFSSpec(rRefNums[i],
                  desktopPB.ioAPPLParID, appName,
                  appSpec);
            if (err != noErr) continue;

               /* found it! */
            return noErr;

        }
        return fnfErr;
    }
```

Note:
This is a fairly simple search method. For your purposes, you may want to modify it so it does special things, such as not bothering with ejected disks that are still mounted, or searching particular disks before others. The code fragment searches local volumes only by checking to see if they are local using the `PBHGetVolParms` routine. In the code snippet in this Technote, local volumes are searched before an optional search of remote volumes takes place.

Launching the App

Once you find the correct application, the next step is to coerce the Apple Event that `AESend` failed to send into the correct format so a pointer to the parameters can be stored in the launch parameter block. This can be achieved with a call to `AECoerceDesc` specifying `typeAppParameters` as the destination type:

```
            /* ask the Apple Event Manager to coerce the Apple
             event into a launch application parameter block
             record. */
        err = AECoerceDesc(&theEvent, typeAppParameters,
              &paraDesc);

             /* copy the record's data into our own memory
             location so it can be incorporated into the
             LaunchApplication parameter block.  Here, the new
             location is referenced using the variable paraData. */
    #if TARGET_API_MAC_CARBON
        paraSize = AEGetDescDataSize(&paraDesc);
        paraData = (AppParametersPtr) NewPtr(paraSize);
        if (paraData == NULL) { err = memFullErr; goto bail; }
        err = AEGetDescData(&paraDesc, paraData, paraSize );
        if (err != noErr) goto bail;
    #else
        paraSize = GetHandleSize((Handle) paraDesc.dataHandle);
        paraData = (AppParametersPtr) NewPtr(paraSize);
        if (paraData == NULL) { err = memFullErr; goto bail; }
        BlockMoveData(*paraDesc.dataHandle, paraData, paraSize);
    #endif
```

Once this coercion has been performed, everything required for setting up the launch parameter block is available so the application can be launched:

```
       launchPB.launchBlockID = extendedBlock;
        launchPB.launchEPBLength = extendedBlockLen;
        launchPB.launchFileFlags = 0;
        launchPB.launchControlFlags =
            launchContinue + launchNoFileFlags;
        launchPB.launchAppSpec = &appSpec;
        launchPB.launchAppParameters = paraData;
        err = LaunchApplication(&launchPB);
```

If the Target Application is Not Apple Event Aware

If the target application is not high-level event aware (pre-System 7), the Apple Event Manager will automatically coerce the event into the old-style launch parameters when the program is being launched. The old-style '`mstr`' and '`mst#`' resources used by MultiFinder in System 6 are no longer supported; therefore, you can't use the routines described herein to open the document if the application is already running.

Back to top

Collected Fragments in a Code Snippet

The following code snippet combines all of the concepts previously explained.

```
       /* Code snippet illustrating how to launch an application
        and send it an open document Apple Event.
        Copyright (C) 1995,2000 by Apple Computer, Inc.
```

```
    All Rights Reserved.*/

#include <Types.h>
#include <Fonts.h>
#include <Files.h>
#include <Errors.h>
#include <Aliases.h>

#include <AppleEvents.h>
#include <AEDataModel.h>


    /* IsRemoteVolume can be used to find out if the
    volume referred to by vRefNum is a remote volume
    located somewhere on a network. the volume's attribute
    flags (copied from the GetVolParmsInfoBuffer structure)
    are returned in the longword pointed to by vMAttrib. */
OSErr IsRemoteVolume(short vRefNum, Boolean *isRemote, long *vMAttrib) {
    HParamBlockRec volPB;
    GetVolParmsInfoBuffer volinfo;
    OSErr err;
    volPB.ioParam.ioVRefNum = vRefNum;
    volPB.ioParam.ioNamePtr = NULL;
    volPB.ioParam.ioBuffer = (Ptr) &volinfo;
    volPB.ioParam.ioReqCount = sizeof(volinfo);
    err = PBHGetVolParmsSync(&volPB);
    if (err == noErr) {
        *isRemote = (volinfo.vMServerAdr != 0);
        *vMAttrib = volinfo.vMAttrib;
    }
    return err;
}


    /* BuildVolumeList fills the array pointed to by vols with
    a list of the currently mounted volumes.  If includeRemote
    is true, then remote server volumes will be included in
    the list.  When remote server volumes are included in the
    list, they will be added to the end of the list.  On entry,
    *count should contain the size of the array pointed to by
    vols.  On exit, *count will be set to the number of id numbers
    placed in the array. If vMAttribMask is non-zero, then
    only volumes with matching attributes are added to the
    list of volumes. bits in the vMAttribMask should use the
    same encoding as bits in the vMAttrib field of
    the GetVolParmsInfoBuffer structure. */
OSErr BuildVolumeList(Boolean includeRemote, short *vols,
        long *count, long vMAttribMask) {
    HParamBlockRec volPB;
    Boolean isRemote;
    OSErr err;
    long nlocal, nremote;

        /* set up and check parameters */
    volPB.volumeParam.ioNamePtr = NULL;
```

```
        nlocal = nremote = 0;
        if (*count == 0) return noErr;

            /* iterate through volumes */
        for (volPB.volumeParam.ioVolIndex = 1;
            PBHGetVInfoSync(&volPB) == noErr;
            volPB.volumeParam.ioVolIndex++) {

                    /* skip remote volumes, if necessary */
            err = IsRemoteVolume(volPB.volumeParam.ioVRefNum, &isRemote);
            if (err != noErr) goto bail;
            if ( ( includeRemote || ! isRemote )
            && (vMAttrib & vMAttribMask) == vMAttribMask ) {

                    /* add local volumes at the front, remote
                    volumes at the end */
                if (isRemote)
                    vols[nlocal + nremote++] = volPB.volumeParam.ioVRefNum;
                else {
                    if (nremote > 0)
                        BlockMoveData(vols+nlocal, vols+nlocal+1,
                            nremote*sizeof(short));
                    vols[nlocal++] = volPB.volumeParam.ioVRefNum;
                }

                    /* list full? */
                if ((nlocal + nremote) >= *count) break;
            }
        }
    bail:
        *count = (nlocal + nremote);
        return err;
    }


        /* FindApplication iterates through mounted volumes
        searching for an application with the given creator
        type.  If includeRemote is true, then remote volumes
        will be searched (after local ones) for an application
        with the creator type. */

    #define kMaxVols 20

    OSErr FindApplication(OSType appCreator, Boolean includeRemote,
            FSSpec *appSpec) {
        short rRefNums[kMaxVols];
        long i, volCount;
        DTPBRec desktopPB;
        Str255 appName;
        OSErr err;

            /* get a list of volumes - with desktop files */
        volCount = kMaxVols;
        err = BuildVolumeList(includeRemote, rRefNums, &volCount,
            (1<<bHasDesktopMgr) );
        if (err != noErr) return err;

            /* iterate through the list */
        for (i=0; i<volCount; i++) {

                /* has a desktop file? */
            desktopPB.ioCompletion = NULL;
            desktopPB.ioVRefNum = rRefNums[i];
            desktopPB.ioNamePtr = NULL;
            desktopPB.ioIndex = 0;
            err = PBDTGetPath(&desktopPB);
            if (err != noErr) continue;

                /* has the correct app?? */
            desktopPB.ioFileCreator = appCreator;
            desktopPB.ioNamePtr = appName;
            err = PBDTGetAPPLSync(&desktopPB);
            if (err != noErr) continue;

                /* make a file spec referring to it */
            err = FSMakeFSSpec(rRefNums[i],
                  desktopPB.ioAPPLParID, appName,
                  appSpec);
            if (err != noErr) continue;

               /* found it! */
            return noErr;

        }
        return fnfErr;
    }




        /* MapCreatorProcs, when provided as a parameter
        to LaunchTheDocument, will be called to allow
        your app to re-map the creator type for a file
        to a different creator type. Your function
        should either modify *creator to refer to a
        preferred application by replacing its value
        with the value of the preferred application's
        creator bytes, or leave *creator untouched if
        you want the creator bytes from the document to
        be used. */

    typedef void (*MapCreatorProc)(FSSpec *document,
        OSType type, OSType *creator);



        /* LaunchTheDocument attempts to open the
        document with either an application that is
        already loaded in memory and running, or by
        finding and launching the appropriate
```

```
        application with the document as a parameter.
        If no application can be found with the creator
        bytes found in the document, fnfErr is returned.
        Local Volumes are searched before remote server
        volumes. Remote server volumes are only searched
        if allowRemoteApps is true. If the optional
        creator remapping procedure is provided
        (remap != NULL), then it will be called to
        allow the caller to re-map the creator bytes
        to a preferred application for the document's
        type.. */

    OSErr LaunchTheDocument(FSSpec *document, Boolean allowRemoteApps,
            MapCreatorProc remap) {
        OSErr err, sresult;
        AEAddressDesc appAddr;
        AEDescList fileList;
        AEDesc fileListElt, paraDesc;
        AppleEvent theEvent;
        AppleEvent eventReply;
        AliasHandle docAlias;
        FInfo fndrInfo;
        OSType appCreator;
        FSSpec appSpec;
        LaunchParamBlockRec launchPB;
        Size paraSize;
        AppParametersPtr paraData;

            /* initialize our records to NULL
            descriptors */
        AECreateDesc(typeNull, NULL, 0, &appAddr);
        AECreateDesc(typeNull, NULL, 0, &fileList);
        AECreateDesc(typeNull, NULL, 0, &fileListElt);
        AECreateDesc(typeNull, NULL, 0, &theEvent);
        AECreateDesc(typeNull, NULL, 0, &eventReply);
        AECreateDesc(typeNull, NULL, 0, &paraDesc);
        docAlias = NULL;
        paraData = NULL;

            /* get the target application's creator */
        err = FSpGetFInfo(document, &fndrInfo);
        if (err != noErr) goto bail;

            /* call the remap function if applicable */
        appCreator = fndrInfo.fdCreator;
        if (remap != NULL)
            remap(document, fndrInfo.fdType,
                &appCreator);

            /* create an open documents Apple Event */
        err = AECreateDesc(typeApplSignature,
              (Ptr)&appCreator, sizeof(OSType), &appAddr);
        if (err != noErr) goto bail;
        err = AECreateAppleEvent(kCoreEventClass, kAEOpenDocuments,
              &appAddr, kAutoGenerateReturnID, kAnyTransactionID,
              &theEvent);
        if (err != noErr) goto bail;

            /* create a one element list of files to send in the
            event */
        err = AECreateList(NULL, 0, false, &fileList);
        if (err != noErr) goto bail;
        err = NewAlias(NULL, document, &docAlias);
        if (err != noErr) goto bail;
        HLock((Handle) docAlias);
        err = AECreateDesc(typeAlias, (Ptr) (*docAlias),
              GetHandleSize((Handle) docAlias), &fileListElt);
        HUnlock((Handle) docAlias);
        if (err != noErr) goto bail;
        err = AEPutDesc(&fileList, 0, &fileListElt);
        if (err != noErr) goto bail;

            /* add the file list to the open documents event */
        err = AEPutParamDesc(&theEvent, keyDirectObject, &fileList);
        if (err != noErr) goto bail;

            /* send the Apple Event */
        err = AESend(&theEvent, &eventReply, kAENoReply,
              kAENormalPriority, kNoTimeOut, NULL, NULL);

            /* if the target could not be found...
            no such app running? */
        if (err == connectionInvalid) {

                /* find an application to launch */
            err = FindApplication(appCreator, allowRemoteApps, &appSpec )
            if (err != noErr) goto bail;

                /* ask the Apple Event Manager to coerce the Apple
                event into a launch application parameter block
                record. */
            err = AECoerceDesc(&theEvent, typeAppParameters,
                  &paraDesc);
            if (err != noErr) goto bail;

                /* copy the record's data into our own memory
                location so it can be incorporated into the
                LaunchApplication parameter block.  Here, the new
                location is referenced using the variable paraData. */
    #if TARGET_API_MAC_CARBON
            paraSize = AEGetDescDataSize(&paraDesc);
            paraData = (AppParametersPtr) NewPtr(paraSize);
            if (paraData == NULL) { err = memFullErr; goto bail; }
            err = AEGetDescData(&paraDesc, paraData, paraSize );
            if (err != noErr) goto bail;
    #else
            paraSize = GetHandleSize((Handle) paraDesc.dataHandle);
            paraData = (AppParametersPtr) NewPtr(paraSize);
            if (paraData == NULL) { err = memFullErr; goto bail; }
            BlockMoveData(*paraDesc.dataHandle, paraData, paraSize);
```

```
#endif

            /* launch the application */
        launchPB.launchBlockID = extendedBlock;
        launchPB.launchEPBLength = extendedBlockLen;
        launchPB.launchFileFlags = 0;
        launchPB.launchControlFlags =
            launchContinue + launchNoFileFlags;
        launchPB.launchAppSpec = &appSpec;
        launchPB.launchAppParameters = paraData;
        err = LaunchApplication(&launchPB);
    }
bail:
        /* clean up, and go.. */
    if (docAlias != NULL) DisposeHandle((Handle) docAlias);
    if (paraData != NULL) DisposePtr((Ptr) paraData);
    AEDisposeDesc(&paraDesc);
    AEDisposeDesc(&appAddr);
    AEDisposeDesc(&fileListElt);
    AEDisposeDesc(&fileList);
    AEDisposeDesc(&theEvent);
    AEDisposeDesc(&eventReply);
    return err;
}


    /* sample creator mapping function. The
    function simply tells LaunchTheDocument
    that every document of type 'TEXT' should
    be opened using MPW Shell */
void MyMapCreator(FSSpec *document, OSType type, OSType *creator) {

        /* open all text files with AppleWorks,
        creator = 'BOBO' */

    if (type == 'TEXT')
        *creator = 'BOBO';
}



void DoTestWithMyDocument(void) {

    FSSpec spec;

    if (FSMakeFSSpec(0, 0, "\pMy Document", &spec) == noErr)

        LaunchTheDocument(&spec, false, MyMapCreator);

}
```

File `SendOpen.r` containing Rez source for the `'SIZE'` resource. Because you're working with Apple Events, you must include a `'SIZE'` resource with the `isHighLevelEventAware` bit set so your application can send and receive Apple

Events. Here is a Rez description of a 'SIZE' resource with this bit turned on:

```
        resource 'SIZE' (-1, purgeable) {
             reserved,
             acceptSuspendResumeEvents,
             reserved,
             canBackground,
             multiFinderAware,
             backgroundAndForeground,
             dontGetFrontClicks,
             ignoreChildDiedEvents,
             is32BitCompatible,
             isHighLevelEventAware,
             localAndRemoteHLEvents,
             isStationeryAware,
             dontUseTextEditServices,
             reserved,
             reserved,
             reserved,
             524288,
             524288
         };
```

Back to top

Asking the Finder to open a document

It is also possible to ask the Finder to find the correct application to open a document. To do this, an application must create an open-documents Apple Event and sent it to the Finder. The following routine illustrates how this can be done.

```
#include <Types.h>
#include <Files.h>
#include <AppleEvents.h>
#include <Errors.h>
#include <AERegistry.h>
#include <Aliases.h>

    /* FinderLaunch converts a list of nTargets FSSpec records
    pointed to by the targetList parameter and converts the
    list to an Apple Event.  It then sends that event to the
    Finder.  The array of FSSpec records pointed to by the
    targetList parameter may contain references to files,
    folders, or applications.  The net effect of this command
    is equivalent to the user selecting an icon in one of the
    Finder's windows and then choosing the open command from
    the Finder's file menu. */
OSErr FinderLaunch(long nTargets, FSSpec *targetList) {
    OSErr err;
    AppleEvent theAEvent, theReply;
    AEAddressDesc fndrAddress;
    AEDescList targetListDesc;
    OSType fndrCreator;
```

```
        Boolean wasChanged;
        AliasHandle targetAlias;
        long index;

            /* set up locals  */
        AECreateDesc(typeNull, NULL, 0, &theAEvent);
        AECreateDesc(typeNull, NULL, 0, &fndrAddress);
        AECreateDesc(typeNull, NULL, 0, &theReply);
        AECreateDesc(typeNull, NULL, 0, &targetListDesc);
        targetAlias = NULL;
        fndrCreator = 'MACS';

            /* verify parameters */
        if ((nTargets == 0) || (targetList == NULL)) {
            err = paramErr;
            goto bail;
        }

            /* create an open documents event targeting the
            finder */
        err = AECreateDesc(typeApplSignature, (Ptr) &fndrCreator,
            sizeof(fndrCreator), &fndrAddress);
        if (err != noErr) goto bail;
        err = AECreateAppleEvent(kCoreEventClass, kAEOpenDocuments,
            &fndrAddress, kAutoGenerateReturnID,
            kAnyTransactionID, &theAEvent);
        if (err != noErr) goto bail;

            /* create the list of files to open */
        err = AECreateList(NULL, 0, false, &targetListDesc);
        if (err != noErr) goto bail;
        for ( index=0; index < nTargets; index++) {
            if (targetAlias == NULL)
                err = NewAlias(NULL, (targetList + index),
                      &targetAlias);
            else err = UpdateAlias(NULL, (targetList + index),
                       targetAlias, &wasChanged);
            if (err != noErr) goto bail;
            HLock((Handle) targetAlias);
            err = AEPutPtr(&targetListDesc, (index + 1),
                  typeAlias, *targetAlias,
                  GetHandleSize((Handle) targetAlias));
            HUnlock((Handle) targetAlias);
            if (err != noErr) goto bail;
        }

            /* add the file list to the Apple Event */
        err = AEPutParamDesc(&theAEvent, keyDirectObject,
              &targetListDesc);
        if (err != noErr) goto bail;

            /* send the event to the Finder */
        err = AESend(&theAEvent, &theReply, kAENoReply,
            kAENormalPriority, kAEDefaultTimeout, NULL, NULL);
            /* clean up and leave */
    bail:
        if (targetAlias != NULL) DisposeHandle((Handle) targetAlias);
        AEDisposeDesc(&targetListDesc);
        AEDisposeDesc(&theAEvent);
        AEDisposeDesc(&fndrAddress);
        AEDisposeDesc(&theReply);
        return err;
    }

    void DoTestWithMyDocument(void) {

        FSSpec spec;

        if (FSMakeFSSpec(0, 0, "\pMy Document", &spec) == noErr)

            FinderLaunch(1, &spec );

    }
```

Asking the Finder to open an a document for you is obviously far more convenient than finding an launching the application yourself. However, it does not allow your application to have the same amount of control over the application selection process as the method described at the beginning of this technote.

Note:
Interestingly enough, if an application sends an open documents event to the Finder that references one or more folders (instead of files), then the Finder will open windows to display the contents of those folders.

Back to top

Summary

Launching an application with a document under System 7 requires the use of the Apple Event Manager to either directly send an open document event to an existing application or indirectly send an open document event to an application as part of its launch sequence. This Technote describes how you can do this in your applications and provides a simple example of how you can use the technique.

Back to top

References

Apple Computer, Inc. (1993) "Creating And Sending Apple Events" (Chapter 5), Inside Macintosh: Interapplication Communication. Addison Wesley.

Apple Computer, Inc. (1993) "Handling the Required Apple Events" (4-11), Inside Macintosh: Interapplication Communication. Addison Wesley.

Apple Computer, Inc. (1993) "Desktop Manager" (Chapter 9), Inside Macintosh: More Macintosh Toolbox. Addison Wesley.

Apple Computer, Inc. (1993) "Obtaining Volume Information" (2-145), Inside Macintosh: Files. Addison Wesley.

Apple Computer, Inc. (1993) "Alias Manager" (Chapter 4), Inside Macintosh: Files. Addison Wesley.

Back to top

Change History

October-15-1995	Created.
July-24-2000	Added Finder section.

Back to top

Downloadables

	Acrobat version of this Note (K)	Download

Back to top

Technical Notes by API | Date | Number | Technology | Title

Developer Documentation | Technical Q&As | Development Kits | Sample Code