Macintosh
Technical Notes

Developer Technical Support

#315: Resolving Alias Files Quietly

Written by: Greg Robbins May 1992

`ResolveAliasFile` always presents the user identity dialog when mounting remote volumes. This Technical Note offers an alternative function, `ResolveAliasFileMountOption`, which uses the previously undocumented `FollowFinderAlias` trap to resolve alias files only if their target is on an already mounted volume. Also included is an `IsAliasFile` routine for identifying alias files.

Introduction

Finder alias files are one aspect of the Macintosh human interface considered "reserved for users." The internal format of Finder alias files is intentionally undefined because it is subject to change and because Finder alias files should be neither created nor altered by applications. The Finder is the user's domain, and Finder alias files are a user convenience.

Most applications do not need to take special steps to accommodate Finder alias files. They are resolved by the Finder before they are passed to an application in an Open Documents Apple event, as well as by the Standard File Package when it creates the reply record. Occasionally an application may need to resolve alias files manually. That is normally done by calling `ResolveAliasFile`, as documented in Chapter 9 of *Inside Macintosh* Volume VI.

If a Finder alias file resolves to an item on an unmounted remote volume, `ResolveAliasFile` will attempt to mount the volume to resolve the alias. For servers, this will bring up the user identity dialog box shown on page 7-24 of *Inside Macintosh* Volume VI. For removable volumes, this will raise the "Please insert…" dialog. Presentation of a dialog may be inappropriate for the application. For example, a Standard File hook procedure that quietly opens the selected file in order to offer a preview of its contents would not want the

dialog presented whenever the user selects an alias to a remote item.

Keeping Quiet

The `ResolveAliasFileMountOption` function listed below allows an application to resolve a Finder alias file only if the alias file's target is on a currently mounted volume. If the target is unavailable and the mountRemoteVols parameter is false, `ResolveAliasFileMountOption` returns the error nsvErr. `ResolveAliasFileMountOption` operates like `ResolveAliasFile` if mountRemoteVols is true. `ResolveAliasFileMountOption` updates the alias file if necessary, and returns fnfErr if the alias file is part of a circular chain.

`ResolveAliasFileMountOption` uses the previously undocumented trap `FollowFinderAlias` to resolve the alias file's `'alis'` resource. This is preferable to passing the `'alis'` to `MatchAlias` because it makes no assumptions about how the alias was created or

how it should be resolved. Finder aliases may actually be relative aliases rather than direct aliases. In any case, FollowFinderAlias will take the steps necessary to resolve them properly.

FollowFinderAlias should not be used except when necessary to resolve the 'alis' resource of a Finder alias file. Aliases created by applications should be resolved with the Alias Manager calls ResolveAlias and MatchAlias.

Note: FollowFinderAlias is used internally by Apple Computer, Inc. It has not been tested for use by application software. While we do not anticipate any problems, it is the responsibility of the developer to ensure that it operates appropriately and reliably for their application.

ResolveAliasFileMountOption uses the function IsAliasFile, also listed below, to determine if a file is an alias file. In keeping with the interface of ResolveAliasFile, IsAliasFile will indicate if the specified item is a folder rather than a file.

Quiet Calls

To determine if an FSSpec points to an alias file, a folder, or neither, use IsAliasFile.

```
FUNCTION IsAliasFile(fileFSSpec: FSSpec;
                     VAR aliasFileFlag: BOOLEAN;
                     VAR folderFlag: BOOLEAN): OSErr;
```

IsAliasFile simply calls PBGetCatInfo to check if the FSSpec's target has its directory or isAlias bits set. These are described in *Inside Macintosh* Volume IV, page 122, and *Inside Macintosh* Volume VI, page 9-36.

To resolve an alias file without any dialogs appearing, use ResolveAliasFileMountOption.

```
FUNCTION ResolveAliasFileMountOption(VAR fileFSSpec: FSSpec;
                                     resolveAliasChains: BOOLEAN;
                                     VAR targetIsFolder: BOOLEAN;
                                     VAR wasAliased: BOOLEAN;
                                     mountRemoteVols: BOOLEAN): OSErr;
```

The first four parameters of ResolveAliasFileMountOption are the same as those of ResolveAliasFile. fileFSSpec is the specification for a file, alias file, or folder. resolveAliasChains should be true if the resolution should follow down a chain of alias files. targetIsFolder is a return parameter that is set if the fileFSSpec points to or resolves to a folder. wasAliased returns true if the input fileFSSpec was for an alias file.

If the mountRemoteVols parameter is true, ResolveAliasFileMountOption will attempt to mount a volume if necessary to resolve an alias file, making the call equivalent to ResolveAliasFile. If mountRemoteVols is false and the file spec is for an alias file that resolves to a volume not currently mounted, the call will return nsvErr rather than attempt to mount it.

The FollowFinderAlias trap is intended only for resolving alias records obtained from Finder alias files.

```
FUNCTION FollowFinderAlias(fromFile: FSSpecPtr;
                           alias: AliasHandle;
                           logon: BOOLEAN;
                           VAR target: FSSpec;
                           VAR wasChanged: BOOLEAN): OSErr;

    INLINE $700F,$A823;  { MOVEQ #$0F,D0; _AliasDispatch; }
```

fromFile is a pointer to a file for a first attempt at a relative resolution; pass a pointer to the alias file's FSSpec for this. alias is a handle to the alias record taken from the alias file's resources. If logon is true, the alias manager will attempt to mount a volume if necessary to complete the resolution. target will be the FSSpec found by the resolution. If wasChanged is true following the call, FollowFinderAlias has updated the alias record, and the caller should call ChangedResource and WriteResource if the updated record is to be saved in the resource file.

FollowFinderAlias does a single resolution; it does not follow a chain of alias files. FollowFinderAlias returns the same errors as MatchAlias.

MPW Pascal

```
{*------------------*
 | IsAliasFile      |
 *------------------*}

FUNCTION IsAliasFile(fileFSSpec: FSSpec;
                     VAR aliasFileFlag: BOOLEAN;
                     VAR folderFlag: BOOLEAN): OSErr;
{ sets aliasFileFlag if the FSSpec points to an alias file;
  sets folderFlag if the FSSpec points to a folder }

  CONST
    kAliasFileBit = 15; { bit of FInfo.fdFlags indicating alias file }
    kDirBit = 4;        { bit of CInfoPBRec.ioFlAttrib indicating directory }

  VAR
    myCInfoPBRec: CInfoPBRec;
    retCode: OSErr;

  BEGIN
    { if called from C we could accidentally be passed nil parameters }
    IF (@fileFSSpec = NIL) OR (@aliasFileFlag = NIL) OR (@folderFlag = NIL) THEN
      BEGIN
        IsAliasFile := paramErr;
        Exit(IsAliasFile);
      END;

    aliasFileFlag := FALSE;
    folderFlag := FALSE;

    { get the item's catalog information }
    WITH myCInfoPBRec DO
      BEGIN
        ioCompletion := NIL;
        ioNamePtr := @fileFSSpec.name;
        ioVRefNum := fileFSSpec.vRefNum;
        ioFDirIndex := 0;
```

```
        ioDirID := fileFSSpec.parID;
        ioFVersNum := 0;  { MFS compatibility; see Tech Note #204 }
      END;
    retCode := PBGetCatInfoSync(@myCInfoPBRec);

    IF retCode = noErr THEN

      { set aliasFileFlag if the item is not a directory and the
        aliasFile bit is set }
      BEGIN
        IF BTst(myCInfoPBRec.ioFlAttrib, kDirBit) THEN
          folderFlag := TRUE
        ELSE IF BTst(myCInfoPBRec.ioFlFndrInfo.fdFlags, kAliasFileBit) THEN
          aliasFileFlag := TRUE;
      END;

    IsAliasFile := retCode;
  END;


{*-----------------------------*
 | ResolveAliasFileMountOption |
 *-----------------------------*}

FUNCTION ResolveAliasFileMountOption(VAR fileFSSpec: FSSpec;
                                     resolveAliasChains: BOOLEAN;
                                     VAR targetIsFolder: BOOLEAN;
                                     VAR wasAliased: BOOLEAN;
                                     mountRemoteVols: BOOLEAN): OSErr;

{ ResolveAliasFileMountOption operates identically to ResolveAliasFile,
  except that if mountRemoteVols is false, no attempt will be made to
  resolve aliases that point to items on non-local volumes }

{ if mountRemoteVols is false, ResolveAliasFileMountOption returns nsvErr if
  fileFSSpec points to an unmounted volume }

{ this routine requires the Alias Manager, available under System 7 }

  CONST
    kAliasFileBit = 15;  { bit of FInfo.fdFlags indicating alias file }
    kMaxChains = 10;  { maximum number of aliases to resolve before giving up }

  VAR
    myResRefNum, chainCount: INTEGER;
    alisHandle: Handle;
    initFSSpec: FSSpec;
    updateFlag, foundFlag, wasAliasedTemp, specChangedFlag: BOOLEAN;
    retCode: OSErr;

  FUNCTION FollowFinderAlias(fromFile: FSSpecPtr;
                             alias: AliasHandle;
                             logon: BOOLEAN;
                             VAR target: FSSpec;
                             VAR wasChanged: BOOLEAN): OSErr;

    INLINE $700F,$A823;  { MOVEQ #$0F,D0; _AliasDispatch; }

  { FollowFinderAlias resolves an alias taken from a Finder alias file,
    updating the alias record (but not the alias resource in the file) if
    necessary.

    Warning: This trap is used internally by Apple Computer, Inc.
      It has not been tested for use by application software.
      While we do not anticipate any problems, it is the responsibility
      of the developer to ensure that it operates appropriately and
```

```
    reliably for their application.

  fromFile is a pointer to a file for a first attempt at a relative search
  (pass the alias file's FSSpec); alias is a handle for the alias record
  taken from the file's resources; the alias manager will attempt to mount
  a volume if logon is TRUE; target is the found FSSpec; wasChanged is set
  to TRUE if the alias record needs updating.

  FollowFinderAlias does a single resolution; it does not follow a chain of
  alias files.

  FollowFinderAlias returns the same errors as MatchAlias. }


BEGIN { ResolveAliasFileMountOption }

  { check parameters }
  IF (@fileFSSpec = NIL) OR (@targetIsFolder = NIL) OR (@wasAliased = NIL) THEN
    BEGIN
      ResolveAliasFileMountOption := paramErr;
      Exit(ResolveAliasFileMountOption);
    END;

  initFSSpec := fileFSSpec; { so FSSpec can be restored in case of error }
  chainCount := kMaxChains; { circular alias chain protection }
  targetIsFolder := FALSE;
  foundFlag := FALSE;
  specChangedFlag := FALSE; { in case of error, restore file spec if it changed }
  myResRefNum := -1;        { resource file not open }

  { loop through chain of alias files }
  REPEAT

    chainCount := chainCount - 1;

    { check if FSSpec is an alias file or a directory }
    { note that targetIsFolder => NOT wasAliased }
    retCode := IsAliasFile(fileFSSpec, wasAliased, targetIsFolder);
    IF (retCode <> noErr) OR (NOT wasAliased) THEN Leave; { break from loop }

    { get the resource file reference number }
    myResRefNum := FSpOpenResFile(fileFSSpec, fsCurPerm);
    retCode := ResError;
    IF myResRefNum = -1 THEN Leave;

    { the first 'alis' resource in the file is the appropriate alias }
    alisHandle := Get1IndResource(rAliasType, 1);
    retCode := ResError;
    IF alisHandle = NIL THEN Leave;

    { load the resource explicitly in case SetResLoad(FALSE) }
    LoadResource(alisHandle);
    retCode := ResError;
    IF retCode <> noErr THEN Leave;

    retCode := FollowFinderAlias(@fileFSSpec, AliasHandle(alisHandle),
      mountRemoteVols, fileFSSpec, updateFlag);
    { FollowFinderAlias returns nsvErr if volume not mounted }

    IF retCode = noErr THEN
      BEGIN
        IF updateFlag THEN
          { the resource in the alias file needs updating }
          BEGIN
            { we don't care if these cause errors, which they may
              do if we don't have write permission }
```

```
              ChangedResource(alisHandle);
              WriteResource(alisHandle);
            END;

          specChangedFlag := TRUE; { in case of error, restore file spec }

          retCode := IsAliasFile(fileFSSpec, wasAliasedTemp, targetIsFolder);

          IF retCode = noErr THEN
            { we're done unless it was an alias file and we're
              following a chain }
            foundFlag := NOT (wasAliasedTemp AND resolveAliasChains);
        END;

      CloseResFile(myResRefNum);
      myResRefNum := -1;

    UNTIL (retCode <> noErr) OR (chainCount = 0) OR (foundFlag);

    { return file not found error for circular alias chains }
    IF (chainCount = 0) AND (NOT foundFlag) THEN retCode := fnfErr;

    { if error occurred, close resource file and restore the original FSSpec }

    IF myResRefNum <> -1 THEN CloseResFile(myResRefNum);
    IF (retCode <> noErr) AND (specChangedFlag) THEN fileFSSpec := initFSSpec;

    ResolveAliasFileMountOption := retCode;
  END;
```

MPW C

```
/*-------------*
 | IsAliasFile |
 *-------------*/

pascal OSErr IsAliasFile(const FSSpec *fileFSSpec,
                         Boolean *aliasFileFlag,
                         Boolean *folderFlag)
/* sets aliasFileFlag if the FSSpec points to an alias file;
   sets folderFlag if the FSSpec points to a folder */

{
  CInfoPBRec myCInfoPBRec;
  OSErr retCode;

  if (fileFSSpec == nil || aliasFileFlag == nil || folderFlag == nil)
    return paramErr;

  *aliasFileFlag = *folderFlag = false;

  /* get the item's catalog information */
  myCInfoPBRec.hFileInfo.ioCompletion = nil;
  myCInfoPBRec.hFileInfo.ioNamePtr = &fileFSSpec->name;
  myCInfoPBRec.hFileInfo.ioVRefNum = fileFSSpec->vRefNum;
  myCInfoPBRec.hFileInfo.ioDirID = fileFSSpec->parID;
  myCInfoPBRec.hFileInfo.ioFVersNum = 0;  /* MFS compatibility, see TN #204 */
  myCInfoPBRec.hFileInfo.ioFDirIndex = 0;

  retCode = PBGetCatInfoSync(&myCInfoPBRec);

  /* set aliasFileFlag if the item is not a directory and the
     aliasFile bit is set */
```

```
  if (retCode == noErr) {
    /* check directory bit */
    if ((myCInfoPBRec.hFileInfo.ioFlAttrib & ioDirMask) != 0)
      *folderFlag = true;

    /* check isAlias bit */
    else if ((myCInfoPBRec.hFileInfo.ioFlFndrInfo.fdFlags & 0x8000) != 0)
      *aliasFileFlag = true;
  }

  return retCode;
}


/*--------------------*
 | FollowFinderAlias |
 *--------------------*/

pascal OSErr FollowFinderAlias(const FSSpec *fromFile,
                               AliasHandle alias,
                               Boolean logon,
                               FSSpec *target,
                               Boolean *wasChanged)

  = {0x700F, 0xA823};  /* MOVEQ #$0F,D0; _AliasDispatch; */

/*  FollowFinderAlias resolves an alias taken from a Finder alias file,
    updating the alias record (but not the alias resource in the file) if
    necessary.

    Warning: This trap is used internally by Apple Computer, Inc.
      It has not been tested for use by application software.
      While we do not anticipate any problems, it is the responsibility
      of the developer to ensure that it operates appropriately and
      reliably for their application. */

/*-----------------------------*
 | ResolveAliasFileMountOption |
 *-----------------------------*/

pascal OSErr ResolveAliasFileMountOption(FSSpec *fileFSSpec,
                                         Boolean resolveAliasChains,
                                         Boolean *targetIsFolder,
                                         Boolean *wasAliased,
                                         Boolean mountRemoteVols)
{
/* maximum number of aliases to resolve before giving up */
#define MAXCHAINS 10

  short myResRefNum;
  Handle alisHandle;
  FSSpec initFSSpec;
  Boolean updateFlag, foundFlag, wasAliasedTemp, specChangedFlag;
  short chainCount;
  OSErr retCode;

  if (fileFSSpec == nil || targetIsFolder == nil || wasAliased == nil)
    return paramErr;

  initFSSpec = *fileFSSpec; /* so FSSpec can be restored in case of error */
  chainCount = MAXCHAINS;   /* circular alias chain protection */
  myResRefNum = -1;         /* resource file not open */

  *targetIsFolder = foundFlag = specChangedFlag = false;
```

```
    /* loop through chain of alias files */
    do {
      chainCount--;

      /* check if FSSpec is an alias file or a directory */
      /* note that targetIsFolder => NOT wasAliased      */

      retCode = IsAliasFile(fileFSSpec, wasAliased, targetIsFolder);
      if (retCode != noErr || !(*wasAliased)) break;

      /* get the resource file reference number */
      myResRefNum = FSpOpenResFile(fileFSSpec, fsCurPerm);
      retCode = ResError();
      if (myResRefNum == -1) break;

      /* the first 'alis' resource in the file is the appropriate alias */
      alisHandle = Get1IndResource(rAliasType, 1);
      retCode = ResError();
      if (alisHandle == nil) break;

      /* load the resource explicitly in case SetResLoad(FALSE) */
      LoadResource(alisHandle);
      retCode = ResError();
      if (retCode != noErr) break;

      retCode = FollowFinderAlias(fileFSSpec, (AliasHandle) alisHandle,
        mountRemoteVols, fileFSSpec, &updateFlag);
      /* FollowFinderAlias returns nsvErr if volume not mounted */

      if (retCode == noErr) {

        if (updateFlag) {
          /* the resource in the alias file needs updating */
          ChangedResource(alisHandle);
          WriteResource(alisHandle);
        }

        specChangedFlag = true; /* in case of error, restore file spec */

        retCode = IsAliasFile(fileFSSpec, &wasAliasedTemp, targetIsFolder);
        if (retCode == noErr)
          /* we're done unless it was an alias file and we're following a chain */
          foundFlag = !(wasAliasedTemp && resolveAliasChains);

      }

      CloseResFile(myResRefNum);
      myResRefNum = -1;

    } while (retCode == noErr && chainCount > 0 && !foundFlag);

    /* return file not found error for circular alias chains */
    if (chainCount == 0 && !foundFlag) retCode = fnfErr;

    /* if error occurred, close resource file and restore the original FSSpec */

    if (myResRefNum != -1) CloseResFile(myResRefNum);
    if (retCode != noErr && specChangedFlag) *fileFSSpec = initFSSpec;

    return retCode;
  }
```

Further Reference:

- *Inside Macintosh*, Volume VI, Alias Manager
- *Inside Macintosh*, Volume VI, Finder Interface, pp. 9-29 to 9-32