This chapter describes how your application can use the Alias Manager to establish and resolve **alias records,** which are data structures that describe file system objects (that is, files, directories, and volumes). You create an alias record to take a "fingerprint" of a file system object, usually a file, that you might need to locate again later. You can store the alias record, instead of a file system specification, and then let the Alias Manager find the file again when it's needed. The Alias Manager contains algorithms for locating files that have been moved, renamed, copied, or restored from backup.

Note
The Alias Manager lets you manage alias records. It does not directly manipulate Finder aliases, which the user creates and manages through the Finder. The chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials* describes Finder aliases and ways to accommodate them in your application. ◆

The Alias Manager is available only in system software version 7.0 or later. Use the `Gestalt` function, described in the chapter "Gestalt Manager" of *Inside Macintosh: Operating System Utilities*, to determine whether the Alias Manager is present.

Read this chapter if you want your application to create and resolve alias records. You might store an alias record, for example, to identify a customized dictionary from within a word-processing document. When the user runs a spelling checker on the document, your application can ask the Alias Manager to resolve the record to find the correct dictionary.

To use this chapter, you should be familiar with the File Manager's conventions for identifying files, directories, and volumes, as described in the chapter "Introduction to File Management" in this book.

This chapter begins with a description of the Alias Manager, alias records, and the search strategies that the Alias Manager uses to resolve an alias record. Then this chapter shows how you can

- create alias records
- resolve alias records
- store alias records as resources
- get information about the target of an alias record

About the Alias Manager

The Alias Manager is the part of the Operating System that helps you to locate specified files, directories, or volumes at a later time. It stores certain information about the **target** object in an alias record. When you later want the Alias Manager to find the target, you pass it the alias record and other information regarding the type of search to perform. Sometimes, the Alias Manager can find the original file. In other cases, the Alias Manager builds a list of potential matches and allows you (or the user) to select the desired file.

The Alias Manager creates and **resolves** (that is, finds the targets of) alias records. In general, you should use the Alias Manager to create an alias record whenever you find yourself storing a specific file description, such as filename and parent directory ID. The Alias Manager stores this information and more in the alias record, and it also provides a set of search strategies for resolving the record later. The search strategies are described in "Search Strategies" beginning on page 4-5. You can use the Alias Manager to create, resolve, and (if necessary) update alias records. You can also obtain information about the target of an alias record without actually resolving the alias record.

The Alias Manager can track files and directories across volumes. If the target of an alias record is on an unmounted AppleShare volume, the Alias Manager automatically mounts the volume when it resolves the alias. If the target object is on an unmounted ejectable volume, the Alias Manager prompts the user to insert the volume.

When the Alias Manager creates an alias record, it allocates the storage, fills in the record, and returns a handle to it. Your application is responsible for storing the record and retrieving it when needed. Your application must also supply strategies for handling various alias-resolution problems, described in "Resolving Alias Records" on page 4-10.

To help you understand and use the Alias Manager, this section provides

- an overview of alias records
- a description of the search strategies the Alias Manager uses to resolve alias records

Alias Records

An alias record is a data structure that describes a file, directory, or volume. The record contains

- location information, such as name and parent directory ID
- verification information, such as creation date, file type, and creator
- volume mounting information (that is, server and zone), if applicable

By storing alias records, you can allow your users to create a robust connection to a file—that is, a connection that can survive the moving or renaming of the target file. The Finder introduced in system software version 7.0, for example, stores alias records in aliases created by the user to represent other files or folders. The Edition Manager uses alias records to support data sharing among separate documents.

An alias record is a reliable way to identify a file system object when your application is communicating with a process that might be running on a different machine.

The creation of an alias record has no effect on the target of the record, except to establish a file ID reference for the target file if one did not previously exist. (See the chapter "File Manager" in this book for a description of file IDs and file ID references.)

The alias record contains only two fields of public information available to your application. The bulk of the record is managed privately by the Alias Manager.

```
TYPE AliasRecord =
   RECORD
      userType:   OSType;        {application's signature}
      aliasSize:  Integer;       {size of record when created}
      {variable-length private data}
   END;
```

Your application can store, in the `userType` field, its own signature or any other data that fits into 4 bytes. When the Alias Manager creates an alias record, it stores 0 in that field.

The Alias Manager stores, in the `aliasSize` field, the size assigned to the record at the time of its creation. Knowing the starting size allows you to store and retrieve data of your own at the end of the record (see "Customizing Alias Records" on page 4-13). An alias record is typically 200 to 300 bytes long.

The private Alias Manager data includes all of the location, verification, and mounting information needed to resolve the alias record with the various search strategies described in this chapter.

Search Strategies

Some of the key features of the Alias Manager are the search strategies built into the alias-resolution functions. The search strategies are designed to find the original target of an alias record, even if the target has been moved, renamed, copied, or restored from backup. Which strategy you use to resolve a particular alias record usually depends on a number of factors, including whether you are willing to sacrifice time to find as many potential targets as possible and whether the target is known to be in a particular volume. This section describes the available search strategies.

You can request either a relative or an absolute search. If you request an absolute search, you can specify whether the search should be either fast or exhaustive. (A relative search is always a fast search.) As you can see, there are three general search strategies available to your application for resolving alias records:

- relative search (always fast)
- absolute fast search
- absolute exhaustive search

The following sections describe these search strategies.

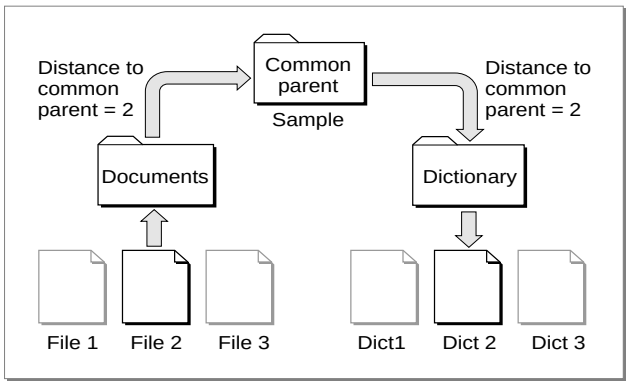
Relative Searches

During a **relative search,** the Alias Manager starts in a specified directory and searches for the target of an alias record by ascending the file system hierarchy to a predetermined common parent of the target and the starting directory and then descending the hierarchy from that common parent.

Suppose, for example, that you are writing a word-processing application that allows the user to build a customized, supplemental dictionary for each document. You might create the dictionary as a separate document in the same directory as the document it serves. In this case, the **common parent** of the document and the dictionary file (that is, the lowest-level directory that appears in the pathnames of both) is simply the directory containing both files.

More generally, you might want to store all document-specific dictionary files in their own directory, as illustrated in Figure 4-1. Here, the common parent of the document file "File 2" and its associated dictionary file "Dict 2" is the directory named "Sample."

Figure 4-1 Resolving a relative path



To resolve an alias record using a relative search, the Alias Manager needs several pieces of information, which are recorded in the alias record at the time you create it. The Alias Manager needs a **relative path,** that is, a path to the target from another file or directory on the same volume. (Relative paths don't work across volumes.) To record a relative path, the Alias Manager saves the distances from the target and the starting file or directory to their common parent. The Alias Manager can later use those distances in conjunction with the full pathname to conduct a relative search.

When resolving the alias record by using a relative path, the Alias Manager looks at the directory at the specified distance above the starting file or directory. The Alias Manager then constructs a partial pathname by extracting one field of the absolute pathname for each step from the target to the common parent. In Figure 4-1, the distance is 2, so the partial pathname is "Dictionary:Dict 2".

Absolute Searches

In contrast to a relative search, an **absolute search** always begins at the root directory of the file system hierarchy and always descends the hierarchy. The first step in any absolute search is to identify the volume on which the target resides. When conducting a volume search, the Alias Manager considers the volume's name, its creation date (which acts almost as a unique identifier for a volume), and its type (for example, a hard disk, a 3.5-inch floppy disk, or an AppleShare volume).

The Alias Manager first looks for a volume that matches all three criteria: name, creation date, and type. The search succeeds if the volume is mounted and if its name and creation date have not changed since the record was created. If the search fails, the Alias Manager attempts to match by creation date and type only. This step locates volumes that have been renamed. Finally, the Alias Manager attempts to match by volume name and type only.

If the target is on an unmounted AppleShare volume, the Alias Manager attempts to mount the volume. It presents a name and password dialog box if appropriate. If the target is on an unmounted ejectable volume, the Alias Manager displays a dialog box prompting the user to insert the volume. Your application can suppress the automatic mounting, as explained in the description of the `MatchAlias` function on page 4-20.

Note

Any time that your application needs to resolve a large number of aliases and the resolution of each alias might require user interaction, you should ensure that if the user cancels any of the dialog boxes, all remaining user interaction is canceled as well. ◆

In some circumstances, a relative search identifies the correct target when an absolute search cannot. For example, suppose the user of your word-processing application creates a working copy of a document and dictionary by copying the entire folder `Sample` to another disk. The user later updates the original document and dictionary by copying the folder from the working disk. All of the underlying file and directory identifications change, but the filenames and relative path remain the same. When the user later runs the spelling checker on the document, a relative-path search finds the correct target dictionary.

Fast Searches

A **fast search** employs an algorithm designed to find the target of an alias record quickly. Depending on how you invoke it, the fast-search algorithm starts with either a relative search or an absolute search. The Alias Manager can perform a relative fast search whether or not it has identified the target volume, but it cannot perform an absolute fast search unless the volume has been identified.

During an absolute fast search, the Alias Manager first searches by file ID (if the target is a file) or directory ID (if the target is a directory). (File IDs and directory IDs are described in the chapter “File Manager” in this book.) Even if a file has been renamed or moved on a volume, the Alias Manager can find it quickly through its file ID.

If the search by file ID or directory ID fails, the Alias Manager searches by name in the original parent directory. This search locates the target if its file or directory ID has changed but it still exists by the same name in the parent directory (for example, if the target was restored from a backup). The Alias Manager compares file numbers of files found by name in the correct parent directory. If the file numbers do not match, the file is treated as a possible match—that is, it is put on the list of candidates—and the search continues. If the target is not found by name in the parent directory, the Alias Manager looks for a file by file number in the parent directory. A file with the same file number but a different name replaces a file with the same name but a different file number in the list of matches.

If the search by file ID or directory ID fails and if the Alias Manager cannot find the original parent directory, it searches for the target by full pathname. This search succeeds if the target resides in the same location on the volume but the directory ID of its parent directory has changed (for example, if the entire parent directory was restored from a backup).

If the search by full pathname fails, the Alias Manager attempts to find the file by tracing partial pathnames up through all parent directories, using parent directory IDs instead of directory names. For example, consider this full pathname:

```
MyDisk:Fruits:Tropical:Ackees
```

If the search by full pathname fails, Alias Manager first looks for the partial pathname ":Ackees" in the directory with the ID that the directory "MyDisk:Fruits:Tropical" had when the alias record was created. If that search fails, it looks for ":Tropical:Ackees" in the directory with the ID that "MyDisk:Fruits" had, and so on.

If you do not ask for a search by relative path first but do provide a starting point for a relative search, and if the alias record contains relative path information, the Alias Manager performs a relative search after the absolute search. The relative search succeeds if the relative path is the same as when the record was created and if the names of the target and its intervening parent directories have not changed.

Exhaustive Searches

An **exhaustive search** uses an algorithm that scans an entire volume to look for possible matches. The Alias Manager typically performs an exhaustive search by calling the File Manager function `PBCatSearch`, searching for files or directories with a matching creation date, creator, and type. (See the chapter "File Manager" in this book for a description of `PBCatSearch`.)

The `PBCatSearch` function is available only on volumes that support the HFS routines and only on systems running system software version 7.0 and later. When `PBCatSearch` is not available, an exhaustive search of the entire volume is performed by making a series of indexed File Manager calls, searching for objects with matching creation date, type, creator, or file number.

Using the Alias Manager

You use the Alias Manager primarily to create and resolve alias records. You can also use it to get information about and update alias records.

The Alias Manager creates an alias record in memory and provides you with a handle to the record. When you no longer need a record in memory, free the memory by calling the Memory Manager's `DisposeHandle` procedure. Whenever possible, you should store and retrieve alias records as resources of type `'alis'`.

Alias Manager functions accept and return file specifications in the form of `FSSpec` records, which contain a volume reference number, a parent directory ID, and a target name. See the chapter "File Manager" in this book for a description of file identification conventions.

Before calling any of the Alias Manager functions, you should verify that the Alias Manager is available by calling the `Gestalt` function with a selector of `gestaltAliasMgrAttr`. If `Gestalt` sets the `gestaltAliasMgrPresent` bit in the response parameter, the Alias Manager is present.

For more detailed descriptions of the functions described in this section, see "Alias Manager Reference" beginning on page 4-13.

Creating Alias Records

You create a new alias record by calling one of three functions: `NewAlias`, `NewAliasMinimal`, or `NewAliasMinimalFromFullPath`. The `NewAlias` function creates a complete alias record that can make full use of the alias-resolution algorithms. The other two functions are streamlined variations designed for circumstances when speed is more important than robust resolution services. All three functions allocate the memory for the record, fill it in, and return a handle to it.

The `NewAlias` function always records the name and the file or directory ID of the target, its creation date, the parent directory name and ID, and the volume name and creation date. It also records the full pathname of the target and a collection of other information. You can have `NewAlias` store relative path information as well by supplying a starting point for a relative path (see "Relative Searches" on page 4-5 for a description of relative paths).

Call `NewAlias` when you want to create an alias record to store for later use. For example, suppose you are writing a word-processing application that allows the user to customize a dictionary for use with a single text file. Your application stores the custom data in a separate dictionary file in the same directory as the document. As soon as you create the dictionary file, you can call `NewAlias` to create an alias record for that file, including path information relative to the user's text file. Listing 4-1 shows how to use `NewAlias` to create a new alias.

Listing 4-1 Creating an alias record

```
FUNCTION DoCreateAlias (myDoc, myDict: FSSpec): OSErr;
VAR
   myAliasHdl: AliasHandle;                    {handle to created alias}
   myErr:      OSErr;
BEGIN
   myErr := NewAlias(@myDoc, myDict, myAliasHdl);  {create alias record}
   IF myAliasHdl <> NIL THEN
      myErr := DoSaveAlias(myDoc, myAliasHdl);     {save it as a resource}
   DoCreateAlias := myErr;                         {return result code}
END;
```

The function `DoCreateAlias` defined in Listing 4-1 takes two `FSSpec` records as parameters. The first specifies the document that is to serve as the starting point for a relative search, in this case the user's text file. The second `FSSpec` record specifies the target of the alias to be created, in this example the dictionary file. The `DoCreateAlias` function calls `NewAlias` to create the alias record; if successful, it calls the application-defined function `DoSaveAlias` to save the alias record as a resource in the document file's resource fork. See Listing 4-2 on page 4-12 for a definition of `DoSaveAlias`.

The two variations on the `NewAlias` function, `NewAliasMinimal` and `NewAliasMinimalFromFullPath`, record only a minimum of information about the target. The `NewAliasMinimal` function records only the target's name, parent directory ID, volume name and creation date, and volume mounting information. The `NewAliasMinimalFromFullPath` function records only the full pathname of the target, including the volume name.

Use `NewAliasMinimal` or `NewAliasMinimalFromFullPath` when you are willing to give up robust alias-resolution service in return for speed. The Finder, for example, stores minimal aliases in the Apple events that tell your application to open or print a document. Because the alias record is resolved almost immediately, the description is likely to remain valid, and the shorter record is probably safe.

You can use `NewAliasMinimalFromFullPath` to create an alias record for a target that doesn't exist or that resides on an unmounted volume.

Resolving Alias Records

The Alias Manager provides two functions that you can use to resolve alias records:

- the high-level function `ResolveAlias`, which performs a fast search and identifies only one target
- the low-level function `MatchAlias`, which can perform a fast search, an exhaustive search, or both and which can return a list of target candidates

In general, when you want to identify only the single most likely target of an alias record, you call `ResolveAlias`. You call `MatchAlias` when you want your program to control the search.

Identifying a Single Target

To resolve an alias record, you usually call the `ResolveAlias` function. This function performs a fast search (described earlier in "Fast Searches" on page 4-7) and exits after it identifies one target. The `ResolveAlias` function compares some key information about the identified target with the information stored in the alias record. If any of the information is different, `ResolveAlias` automatically updates the record.

Note

Like all other Alias Manager functions, `ResolveAlias` updates the record only in memory. Your application is responsible for updating alias records stored on disk when appropriate. ◆

In the dictionary example illustrated in Figure 4-1 on page 4-6, the application calls `ResolveAlias` with a relative path specification when the user runs the spelling checker on a document with a customized dictionary. If you provide a relative starting point, `ResolveAlias` performs the relative search first.

The `ResolveAlias` function reports, in the `wasChanged` parameter, whether it updated the alias record. After `ResolveAlias` runs, the value of `wasChanged` is `TRUE` if the record was updated and `FALSE` if it was not. If you are storing the alias record, check the value of `wasChanged` (as well as the function's result code) to see whether to update the stored record after resolving an alias.

If `ResolveAlias` can't resolve the alias record, it returns a nonzero result code. A result code of `fnfErr` signals that `ResolveAlias` has found the correct volume and parent directory but not the target file or folder. In this case, `ResolveAlias` constructs a valid `FSSpec` record that describes the target. You can use this record to explore possible solutions to the resolution failure. You can, for example, pass the `FSSpec` record to the File Manager function `FSpCreate` to create a replacement for a missing file.

Identifying Multiple Targets

The `MatchAlias` function is a low-level routine that gives your application control over the search algorithms.

You can control

- whether to attempt an automatic mounting of unmounted volumes
- whether to search on more than one volume
- whether to perform a fast search, an exhaustive search, or both
- what the order of the absolute and relative searches in a fast search should be
- whether to pursue search strategies that require interaction with the user (such as asking for a password while mounting an AppleShare volume)

You can also specify a maximum number of candidates that `MatchAlias` can identify. For details about controlling a search with the `MatchAlias` function, see its description beginning on page 4-20.

You can supply an optional filter function that `MatchAlias` calls

- each time it identifies a possible match
- when three seconds have elapsed without a match

The filter function determines whether each candidate is added to the list of possible targets. It can also terminate the search. See "Filtering Possible Targets" on page 4-25 for a description of the filter function.

The `MatchAlias` function returns, in an array of file system specification records, all candidates that it identifies.

Maintaining Alias Records

You can store alias records as resources of type `'alis'`.

```
CONST
   rAliasType = 'alis'; {resource type for saved alias records}
```

To store and retrieve resources, use the standard Resource Manager functions (`AddResource`, `GetResource`, and `GetNamedResource`) described in the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox*. Listing 4-2 illustrates one way to save an alias record as a resource in a document file's resource fork.

Listing 4-2 Storing an alias record as a resource

```
FUNCTION DoSaveAlias (myDoc: FSSpec; myAliasHdl: AliasHandle): OSErr;
VAR
   myErr: OSErr;
   myFile: Integer;           {file ref number of document's resource fork}
CONST
   kID = 129;
   kName = 'Dictionary Alias';
BEGIN
   myFile := FSpOpenResFile(myDoc, fsCurPerm);
   IF myFile = -1 THEN        {couldn't open the document's resource fork}
      BEGIN
         DoSaveAlias := ResError;
         exit(DoSaveAlias);
      END;
   AddResource(Handle(myAliasHdl), rAliasType, kID, kName);
   myErr := ResError;         {check for errors adding resource}
   IF myErr = noErr THEN
      BEGIN
         WriteResource(Handle(myAliasHdl));
         myErr := ResError;   {check for errors writing resource}
      END;
   DoSaveAlias := myErr;
END;
```

Note that `DoSaveAlias` assumes that the file specified by the `myDoc` parameter already has a resource fork and that the file is not yet open. Your application might have different requirements.

To update an alias record, use the `UpdateAlias` function. You typically call `UpdateAlias` any time you know that the target of an alias record has been renamed or otherwise changed. You are most likely to call `UpdateAlias` after a call to the

`MatchAlias` function. If `MatchAlias` identifies a single target, it sets a flag telling you whether or not the key information about the target file matches the information in the alias record. It is the responsibility of your application to update the record.

The `ResolveAlias` function automatically updates an alias record if any of the key information about the identified target does not match the information in the record.

Getting Information From Alias Records

To retrieve information from an alias record without actually resolving the record, call the `GetAliasInfo` function. You can use `GetAliasInfo` to retrieve the name of the target, the names of the target's parent directories, the name of the target's volume, or, in the case of an AppleShare volume, its zone or server name.

The information returned by `GetAliasInfo` might be stale. `GetAliasInfo` reads the information stored in the alias record, which might have changed since the creation of the record. Because it doesn't resolve the alias record, `GetAliasInfo` is most useful for providing information quickly.

Customizing Alias Records

An alias record contains two kinds of information: public information available to your application and private information available only to the Alias Manager. Your application can use the first field, `userType`, to store its own signature or any other data that fits into 4 bytes. Your application can use the second field, `aliasSize`, to customize the alias record for storing additional data.

The Alias Manager stores, in the `aliasSize` field, the size of the record at the time it is created or updated. To customize an alias record, you first use the Memory Manager's `SetHandleSize` procedure to increase the size of the record. You can then find the starting address of your own data in the record by adding the record's starting address to the length recorded in the `aliasSize` field. If you use the Memory Manager to expand the record, the Alias Manager preserves your data, even if it changes the size of its own data when updating the record.

Note
In general, you should customize only alias records that you have created. ◆

Alias Manager Reference

This section describes the routines provided by the Alias Manager and the `AliasRecord` data structure you must pass when calling those routines.

Data Structures

The Alias Manager uses alias records to store information that allows it to locate an object in the file system.

Alias Records

Alias records are defined by the `AliasRecord` data type.

```
TYPE AliasRecord  =
RECORD
   userType:   OSType;        {application's signature}
   aliasSize:  Integer;       {size of record when created}
   {variable-length private data}
END;
```

Field descriptions

`userType` A 4-byte field that can contain application-specific data. When an alias record is created, this field contains 0. Your application can use this field for its own purposes. Typically you should store your application's signature here.

`aliasSize` The size, in bytes, assigned to the alias record at the time of its creation or updating. This is the total size of the record, including the `userType` and `aliasSize` fields, as well as the variable-length data that is private to the Alias Manager.

Following these two fields is a variable-length block of data maintained privately by the Alias Manager.

Alias Manager Routines

This section describes the routines you use to create, update, resolve, and read alias records. Alias Manager routines use file system specification records (defined by the `FSSpec` data type) to identify files, directories, and volumes. To create an `FSSpec` record, call the function `FSMakeFSSpec`, described in the chapter "File Manager" in this book.

The Alias Manager routines can return the result codes listed in this section or any other applicable file system or memory management result codes.

Creating and Updating Alias Records

You can use the functions `NewAlias`, `NewAliasMinimal`, `NewAliasMinimalFromFullPath`, and `UpdateAlias` to create and update alias records.

NewAlias

You use the `NewAlias` function to create a complete alias record.

```
FUNCTION NewAlias (fromFile: FSSpecPtr; target: FSSpec;
                   VAR alias: AliasHandle): OSErr;
```

`fromFile` The starting point for a relative path, to be used later in a relative search. If you do not need relative path information in the record, pass a `fromFile` value of `NIL`. If you want `NewAlias` to record relative path information, pass a pointer to a valid `FSSpec` record in this parameter. The two files or directories, `fromFile` and `target`, must reside on the same volume.

`target` An `FSSpec` record for the target of the alias record.

`alias` A handle to the newly created alias record. If the function fails to create an alias record, it sets `alias` to `NIL`.

DESCRIPTION

The `NewAlias` function creates an alias record that describes the specified target. It allocates the storage, fills in the record, and puts a record handle to that storage in the `alias` parameter. `NewAlias` always records the name and file or directory ID of the target, its creation date, the parent directory name and ID, and the volume name and creation date. It also records the full pathname of the target and a collection of other information relevant to locating the target, verifying the target, and mounting the target's volume, if necessary. You can have `NewAlias` store relative path information as well by supplying a starting point for a relative path (see "Relative Searches" on page 4-5 for a description of relative paths).

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for `NewAlias` are

Trap macro	Selector
`_AliasDispatch`	$0002

RESULT CODE

`noErr`	0	No error

NewAliasMinimal

You use the `NewAliasMinimal` function to create a short alias record quickly.

```
FUNCTION NewAliasMinimal (target: FSSpec;
                          VAR alias: AliasHandle): OSErr;
```

`target` An `FSSpec` record for the target of the alias record.

`alias` A handle to the newly created alias record. If the function fails to create an alias record, it sets `alias` to `NIL`.

DESCRIPTION

The `NewAliasMinimal` function creates an alias record that contains only the minimum information necessary to describe the target: the target name, the parent directory ID, the volume name and creation date, and the volume mounting information. The `NewAliasMinimal` function uses the standard alias record data structure, but it fills in only parts of the record.

Note
The `ResolveAlias` function, described on page 4-19, never updates a minimal alias record. ◆

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for `NewAliasMinimal` are

Trap macro	Selector
`_AliasDispatch`	$0008

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	The value of `target` or `alias` parameter, or of both, is `NIL`, or the alias record is corrupt

NewAliasMinimalFromFullPath

You use the function `NewAliasMinimalFromFullPath` to quickly create an alias record that contains only the full pathname of the target.

```
FUNCTION NewAliasMinimalFromFullPath
                  (fullPathLength: Integer; fullPath: Ptr;
                     zoneName: Str32; serverName: Str31;
                     VAR alias: AliasHandle): OSErr;
```

`fullPathLength`
The number of characters in the full pathname of the target.

`fullPath` A pointer to a buffer that contains the full pathname of the target. The full pathname starts with the name of the volume, includes all of the directory names in the path to the target, and ends with the target name. (For a description of pathnames, see the chapter "File Manager" in this book.)

`zoneName` The AppleTalk zone name of the AppleShare volume on which the target resides. Set this parameter to a null string if you do not need it.

`serverName` The AppleTalk server name of the AppleShare volume on which the target resides. Set this parameter to a null string if you do not need it.

`alias` A handle to the newly created alias record. If the function fails to create an alias record, it sets `alias` to `NIL`.

DESCRIPTION

The `NewAliasMinimalFromFullPath` function creates an alias record that identifies the target by full pathname. You can call `NewAliasMinimalFromFullPath` to create an alias record for a file that doesn't exist or that resides on an unmounted volume.

The `NewAliasMinimalFromFullPath` function uses the standard alias record data structure, but it fills in only the information provided in the input parameters. You can therefore use `NewAliasMinimalFromFullPath` to create alias records for targets on unmounted volumes.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for `NewAliasMinimalFromFullPath` are

Trap macro	Selector
`_AliasDispatch`	$0009

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Parameter error

UpdateAlias

You use the `UpdateAlias` function to update an alias record.

```
FUNCTION UpdateAlias (fromFile: FSSpecPtr; target: FSSpec;
                      alias: AliasHandle;
                      VAR wasChanged: Boolean): OSErr;
```

`fromFile` The starting point for a relative path, to be used later in a relative search. If you do not need relative path information in the record, pass a `fromFile` value of `NIL`. If you want `UpdateAlias` to record relative path information, pass a pointer to a valid `FSSpec` record in this parameter.

`target` The target of the alias record. This parameter must be a valid `FSSpec` record.

`alias` A handle to the alias record to be updated.

`wasChanged` A Boolean value indicating whether the newly constructed alias record is exactly the same as the old one. If the new record is the same as the old one, `UpdateAlias` sets the `wasChanged` parameter to `FALSE`. Otherwise, it sets it to `TRUE`. Check this parameter to determine whether you need to save an updated record.

DESCRIPTION

The `UpdateAlias` function updates the alias record pointed to by the `alias` parameter so that it describes the target specified by the `target` parameter. The `UpdateAlias` function rebuilds the entire alias record and fills it in as the `NewAlias` function would.

The `UpdateAlias` function always creates a complete alias record. When you use `UpdateAlias` to update a minimal alias record, you convert the minimal record to a complete record.

SPECIAL CONSIDERATIONS

The two files or directories, `fromFile` and `target`, must reside on the same volume.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for `UpdateAlias` are

Trap macro	Selector
`_AliasDispatch`	$0006

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	The value of the `target` or `alias` parameter, or both, is `NIL`, or the alias record is corrupt

Resolving and Reading Alias Records

You can use the functions `ResolveAlias` and `MatchAlias` to resolve or find possible targets of an alias record. You can use the function `GetAliasInfo` to get information about the target of an alias without actually resolving the alias.

ResolveAlias

You use the `ResolveAlias` function to identify the single most likely target of an alias record.

```
FUNCTION ResolveAlias (fromFile: FSSpecPtr; alias: AliasHandle;
                       VAR target: FSSpec;
                       VAR wasChanged: Boolean): OSErr;
```

`fromFile` The starting point for a relative search. If you pass a `fromFile` parameter of `NIL`, `ResolveAlias` performs only an absolute search. If you pass a pointer to a valid `FSSpec` record in the `fromFile` parameter, `ResolveAlias` performs a relative search for the target, followed by an absolute search only if the relative search fails. If you want to perform an absolute search followed by a relative search, you must use the `MatchAlias` function.

`alias` A handle to the alias record to be resolved and, if necessary, updated.

`target` The target of the alias record. This parameter must be a valid `FSSpec` record.

`wasChanged` A Boolean value indicating whether the alias record to be resolved was updated because it contained some outdated information about the target.

DESCRIPTION

The `ResolveAlias` function performs a fast search for the target of the alias, as described in "Fast Searches" on page 4-7. If the resolution is successful, `ResolveAlias` returns (in the `target` parameter) the `FSSpec` record for the target file system object, updates the alias record if necessary, and reports (through the `wasChanged` parameter) whether the record was updated. If the target is on an unmounted AppleShare volume, `ResolveAlias` automatically mounts the volume. If the target is on an unmounted ejectable volume, `ResolveAlias` asks the user to insert the volume. The `ResolveAlias` function exits after it finds one acceptable target.

After it identifies a target, `ResolveAlias` compares some key information about the target with the information in the alias record. (The description of the `MatchAlias` function, beginning on page 4-20, lists the key information.) If the information differs, `ResolveAlias` updates the record to match the target. If it updates the alias record, `ResolveAlias` sets the `wasChanged` parameter to `TRUE`. Otherwise, it sets it to `FALSE`. (`ResolveAlias` never updates a minimal alias, so it never sets `wasChanged` to `TRUE` when resolving a minimal alias.)

When it finds the specified volume and parent directory but fails to find the target file or directory in that location, `ResolveAlias` returns a result code of `fnfErr` and fills in the `target` parameter with a complete `FSSpec` record describing the target (that is, the volume reference number, parent directory ID, and filename or folder name). The `FSSpec` record is valid, although the object it describes does not exist. This information is intended as a "hint" that lets you explore possible solutions to the resolution failure. You can, for example, pass the `FSSpec` record to the File Manager function `FSpCreate` to create a replacement for a missing file.

The `ResolveAlias` function displays the standard dialog boxes when it needs input from the user, such as a name and password for mounting a remote volume. The user can cancel the resolution through these dialog boxes.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for `ResolveAlias` are

Trap macro	Selector
`_AliasDispatch`	$0003

RESULT CODES

`noErr`	0	No error
`nsvErr`	–35	The volume is not mounted
`fnfErr`	–43	Target not found, but volume and parent directory found
`paramErr`	–50	The value of the `target` or `alias` parameter, or both, is `NIL`, or the alias record is corrupt
`dirNFErr`	–120	Parent directory not found
`usrCanceledErr`	–128	The user canceled the operation

MatchAlias

You use the `MatchAlias` function to identify a list of possible matches and pass the list through an optional selection filter. The filter can return more than one possible match.

```
FUNCTION MatchAlias (fromFile: FSSpecPtr; rulesMask: LongInt;
                     alias: AliasHandle; VAR aliasCount: Integer;
                     aliasList: FSSpecArrayPtr;
                     VAR needsUpdate: Boolean;
                     aliasFilter: AliasFilterProcPtr;
                     yourDataPtr: UNIV Ptr): OSErr;
```

`fromFile` The starting point for a relative search. If you do not want `MatchAlias` to perform a relative search, set `fromFile` to `NIL`. If you want `MatchAlias` to perform a relative search, pass a pointer to a file system specification record that describes the starting point for the search.

`rulesMask` A set of rules to guide the resolution. Pass the sum of all of the rules you want to invoke.

`alias` A handle to the alias record to be resolved.

`aliasCount` On input, the maximum number of possible matches to return. On output, the actual number of matches returned.

`aliasList` A pointer to the array that holds the results of the search.

`needsUpdate`
A Boolean flag that indicates whether the alias record to be resolved needs to be updated.

`aliasFilter`
An application-defined filter function.

`yourDataPtr`
A pointer to data to be passed to the filter function.

DESCRIPTION

The `MatchAlias` function resolves the alias record specified by the `alias` parameter, following the rules specified by the `rulesMask` parameter. Then it returns, in the structure specified by the `aliasList` parameter, a list of possible candidates. The `MatchAlias` function places, in the `aliasCount` parameter, the number of candidates identified.

You specify the matching criteria by passing a sum of these constants in the `rulesMask` parameter.

```
CONST
   kARMMountVol         =  $00000001;{mount volume automatically}
   kARMNoUI             =  $00000002;{suppress user interface}
   kARMMultVols         =  $00000008;{search on multiple volumes}
   kARMSearch           =  $00000100;{do a fast search}
   kARMSearchMore       =  $00000200;{do an exhaustive search}
   kARMSearchRelFirst   =  $00000400;{do a relative search first}
```

Constant descriptions

`kARMMountVol` Automatically try to mount the target's volume if it is not mounted.

`kARMNoUI` Stop if a search requires user interaction, such as a password dialog box when mounting a remote volume. If user interaction is needed and `kARMNoUI` is in effect, the search fails.

`kARMMultVols` Search all mounted volumes. The search begins with the volume on which the target resided when the record was created. When you specify a fast search of all mounted volumes, `MatchAlias` performs a formal fast search only on the volume described in the alias record. On all other volumes it looks for the target by ID or by name in the directory with the specified parent directory ID. When you specify an exhaustive search of multiple volumes, `MatchAlias` performs the same search on all volumes. When resolving an alias record created by `NewAliasMinimalFromFullPath`, `MatchAlias` ignores this flag.

`kARMSearch` Perform a fast search for the alias target. If `kARMSearchRelFirst` is not set, perform an absolute search first, followed by a relative search only if the value of the `fromFile` parameter is not `NIL` and the list of matches is not full.

`kARMSearchMore` Perform an exhaustive search for the alias target. On HFS volumes, the exhaustive search uses the File Manager function `PBCatSearch` to identify candidates with matching creation date, type, and creator. The `PBCatSearch` function is available only on HFS volumes and only on systems running version 7.0 or later. On MFS volumes or HFS volumes that do not support `PBCatSearch`, the exhaustive search makes a series of indexed calls to File Manager functions, using the same search criteria. If you set `kARMSearchMore` and either or both of `kARMSearch` and `kARMSearchRelFirst`, `MatchAlias` performs the fast search first.

`kARMSearchRelFirst`
If `kARMSearch` is also set, perform a relative search before the absolute search. (If `kARMSearch` is also set and the target is found through the absolute search, `MatchAlias` sets the `needsUpdate` flag to `TRUE`.) If neither `kARMSearch` nor `kARMSearchMore` is set, perform only a relative search. If `kARMSearch` is not set but `kARMSearchMore` is set, perform a relative search followed by an exhaustive search.

You must specify at least one of the last three parameters: `kARMSearch`, `kARMSearchMore`, and `kARMSearchRelFirst`.

Your application can specify a maximum number of possible matches by setting the `aliasCount` parameter. `MatchAlias` changes the `aliasCount` parameter to the actual number of candidates identified. If `MatchAlias` finds the parent directory on the correct volume but does not find the target, it sets the `aliasCount` parameter to 1, puts the file system specification record for the target in the results list, and returns `fnfErr`. The `FSSpec` record is valid, although the object it describes does not exist. This information is intended as a "hint" that lets you explore possible solutions to the resolution failure. You can, for example, use the `FSSpec` record and the File Manager function `FSpCreate` to create a replacement for a missing file.

The `needsUpdate` flag is a signal to your application that the record might need to be updated. After it identifies a target, `MatchAlias` compares some key information about the target with the same information in the record. If the information does not match, `MatchAlias` sets the `needsUpdate` flag to `TRUE`. The key information is

- the name of the target
- the directory ID of the target's parent
- the file ID or directory ID of the target
- the name and creation date of the volume on which the target resides

The `MatchAlias` function also sets the `needsUpdate` flag to `TRUE` if it identifies a list of possible matches rather than a single match or if `kARMsearchRelFirst` is set but the target is identified through either an absolute search or an exhaustive search. Otherwise, the `MatchAlias` function sets the `needsUpdate` flag to `FALSE`. `MatchAlias` always

sets the `needsUpdate` flag to `FALSE` when resolving an alias created by `NewAliasMinimal`. If you want to update the alias record to reflect the final results of the resolution, call `UpdateAlias`.

The `aliasFilter` parameter points to a filter function supplied by your application. The Alias Manager executes this function each time it identifies a possible match and after the search has continued for three seconds without a match. Your filter function returns a Boolean value that determines whether the possible match is discarded (`TRUE`) or added to the list of possible targets (`FALSE`). It can also terminate the search by setting the variable parameter `quitFlag`. See "Filtering Possible Targets" on page 4-25 for a description of the filter function.

The `yourDataPtr` parameter can point to any data that your application might need in the filter function.

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for `MatchAlias` are

Trap macro	Selector
`_AliasDispatch`	$0005

RESULT CODES

`noErr`	0	No error
`nsvErr`	–35	The volume is not mounted
`fnfErr`	–43	Target not found, but volume and parent directory found
`paramErr`	–50	The value of the `target` or `alias` parameter, or both, is `NIL`, or the alias record is corrupt
`usrCanceledErr`	–128	The user canceled the operation

GetAliasInfo

You use the `GetAliasInfo` function to get information from an alias record without actually resolving the record.

```
FUNCTION GetAliasInfo (alias: AliasHandle; index: AliasInfoType;
                       VAR theString: Str63): OSErr;
```

`alias`	A handle to the alias record to be read.
`index`	The kind of information to be retrieved.
`theString`	A string that holds the requested information.

DESCRIPTION

The `GetAliasInfo` function retrieves the information specified by the `index` parameter from the record pointed to by the `alias` parameter and places that information in the parameter `theString`.

The `index` parameter specifies the kind of information to be retrieved. If the value of `index` is a positive integer, `GetAliasInfo` retrieves the parent directory that has the same hierarchical level above the target as the index parameter (for example, an `index` value of 2 returns the name of the parent directory of the target's parent directory). You can therefore assemble the names of the target and all of its parent directories by making repeated calls to `GetAliasInfo` with incrementing index values, starting with a value of 0. When the value of `index` is greater than the number of levels between the target and the root, `GetAliasInfo` returns an empty string. You can also set the `index` parameter to one of the following five values:

```
CONST
   asiZoneName          =  -3;      {get zone name}
   asiServerName        =  -2;      {get server name}
   asiVolumeName        =  -1;      {get volume name}
   asiAliasName         =  0;       {get target name}
   asiParentName        =  1;       {get parent directory name}
```

Constant descriptions

`asiZoneName`	If the record represents a target on an AppleShare volume, retrieve the server's zone name. Otherwise, return an empty string.
`asiServerName`	If the record represents a target on an AppleShare volume, retrieve the server name. Otherwise, return an empty string.
`asiVolumeName`	Return the name of the volume on which the target resides.
`asiAliasName`	Return the name of the target.
`asiParentName`	Return the name of the parent directory of the target of the record. If the target is a volume, return the volume name.

The `GetAliasInfo` function returns the information stored in the alias record, which might not be current. To ensure that the information is current, you can resolve and update the alias record before calling `GetAliasInfo`.

Note

The `GetAliasInfo` function cannot provide all kinds of information about a minimal alias. ◆

ASSEMBLY-LANGUAGE INFORMATION

The trap macro and routine selector for `GetAliasInfo` are

Trap macro	Selector
`_AliasDispatch`	$0007

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	The value of `alias` or `theString` parameter, or both, is `NIL`; the value of `index` is less than the value of `asiZoneName`; or the alias record is corrupt

Application-Defined Routines

The Alias Manager supports a single application-defined routine, a function for filtering out possible targets of an alias record.

Filtering Possible Targets

You can write your own filter function to examine possible targets identified by the `MatchAlias` function. The `MatchAlias` function calls your filter function each time it identifies a possible match or when three seconds have elapsed without a match.

MyMatchAliasFilter

You can pass the address of an alias-matching filter function to the `MatchAlias` function.

```
FUNCTION MyMatchAliasFilter (cpbPtr: CInfoPBPtr;
                             VAR quitFlag: Boolean;
                             myDataPtr: Ptr): Boolean;
```

`cpbPtr`	A pointer to a catalog information parameter block.
`quitFlag`	On exit, set this to `TRUE` if you want to terminate the search.
`myDataPtr`	A pointer to custom data.

DESCRIPTION

Your application-defined filter function is called by `MatchAlias` to filter out possible matches. When your function is called, the `cpbPtr` parameter points to the catalog information parameter block of the possible match (returned by the File Manager function `PBGetCatInfo`). The `MatchAlias` function sets this parameter to `NIL` if it is calling your function to give it the periodic chance to terminate the search. (Do not use this pointer without checking for `NIL`.) If you want to terminate the search, set the `quitFlag` parameter to `TRUE`.

The `myDataPtr` parameter points to any customized data that your application passed when it called `MatchAlias`. This parameter allows your filter function to access any data that your application has set up on its own.

Your function should return `TRUE` to indicate that the possible match is to be discarded, or `FALSE` to indicate that the possible match is to be added to the list of possible targets.

Summary of the Alias Manager

Pascal Summary

Constants

```
CONST
   {Gestalt constants}
   gestaltAliasMgrAttr     = 'alis';      {Alias Mgr attributes selector}
   gestaltAliasMgrPresent  = 0;           {Alias Mgr is present}

   {resource type for saved alias records}
   rAliasType              = 'alis';

   {masks for alias resolution action rules used by MatchAlias}
   kARMMountVol            =  $00000001;  {mount volume automatically}
   kARMNoUI                =  $00000002;  {suppress user interface}
   kARMMultVols            =  $00000008;  {search on multiple volumes}
   kARMSearch              =  $00000100;  {do a fast search}
   kARMSearchMore          =  $00000200;  {do an exhaustive search}
   kARMSearchRelFirst      =  $00000400;  {do a relative search first}

   {index values for GetAliasInfo}
   asiZoneName             =  –3;         {get zone name}
   asiServerName           =  –2;         {get server name}
   asiVolumeName           =  –1;         {get volume name}
   asiAliasName            =  0;          {get target name}
   asiParentName           =  1;          {get parent directory name}
```

Data Types

```
TYPE AliasRecord     =                {alias record}
   RECORD
      userType:   OSType;             {application's signature}
      aliasSize:  Integer;            {size of record when created}
      {variable-length private data}
   END;
```

```
AliasPtr            =  ^AliasRecord;
AliasHandle         =  ^AliasPtr;

AliasInfoType       =  Integer;    {alias record information type}

AliasFilterProcPtr  =  ProcPtr;    {application-defined routine}
```

Alias Manager Routines

Creating and Updating Alias Records

```
FUNCTION NewAlias           (fromFile: FSSpecPtr; target: FSSpec;
                             VAR alias: AliasHandle): OSErr;
FUNCTION NewAliasMinimal    (target: FSSpec;
                             VAR alias: AliasHandle): OSErr;
FUNCTION NewAliasMinimalFromFullPath
                            (fullPathLength: Integer; fullPath: Ptr;
                             zoneName: Str32; serverName: Str31;
                             VAR alias: AliasHandle): OSErr;
FUNCTION UpdateAlias        (fromFile: FSSpecPtr; target: FSSpec;
                             alias: AliasHandle;
                             VAR wasChanged: Boolean): OSErr;
```

Resolving and Reading Alias Records

```
FUNCTION ResolveAlias       (fromFile: FSSpecPtr; alias: AliasHandle;
                             VAR target: FSSpec;
                             VAR wasChanged: Boolean): OSErr;
FUNCTION MatchAlias         (fromFile: FSSpecPtr; rulesMask: LongInt;
                             alias: AliasHandle; VAR aliasCount: Integer;
                             aliasList: FSSpecArrayPtr;
                             VAR needsUpdate: Boolean;
                             aliasFilter: AliasFilterProcPtr;
                             yourDataPtr: UNIV Ptr): OSErr;
FUNCTION GetAliasInfo       (alias: AliasHandle; index: AliasInfoType;
                             VAR theString: Str63): OSErr;
```

Application-Defined Routine

```
FUNCTION MyMatchAliasFilter (cpbPtr: CInfoPBPtr; VAR quitFlag: Boolean;
                             myDataPtr: Ptr): Boolean;
```

C Summary

Constants

```
/*Gestalt constants*/
#define gestaltAliasMgrAttr   'alis'      /*Alias Mgr attributes selector*/
#define gestaltAliasMgrPresent 0          /*Alias Mgr is present*/

/*resource type for saved alias records*/
#define rAliasType            'alis'

/*masks for alias resolution action rules used by MatchAlias*/
enum {kARMMountVol      = 0x00000001};    /*mount volume automatically*/
enum {kARMNoUI          = 0x00000002};    /*suppress user interface*/
enum {kARMMultVols      = 0x00000008};    /*search on multiple volumes*/
enum {kARMSearch        = 0x00000100};    /*do a fast search*/
enum {kARMSearchMore    = 0x00000200};    /*do an exhaustive search*/
enum {kARMSearchRelFirt = 0x00000400};    /*do a relative search first*/

/*index values for GetAliasInfo*/
enum {asiZoneName       = –3};            /*get zone name*/
enum {asiServerName     = –2};            /*get server name*/
enum {asiVolumeName     = –1};            /*get volume name*/
enum {asiAliasName      = 0};             /*get target name*/
enum {asiParentName     = 1};             /*get parent directory name*/
```

Data Types

```
typedef struct {                          /*alias record*/
      OSType            userType;         /*application's signature*/
      unsigned short    aliasSize;        /*size of record when created*/
} AliasRecord;

typedef AliasRecord *AliasPtr;
typedef AliasRecord **AliasHandle;
typedef short AliasInfoType;              /*alias record information type*/

typedef pascal Boolean (*AliasFilterProcPtr)(CInfoPBPtr cpbPtr,
                        Boolean *quitFlag, Ptr yourDataPtr);
```

Alias Manager Routines

Creating and Updating Alias Records

```
pascal OSErr NewAlias       (const FSSpec *fromFile, const FSSpec *target,
                             AliasHandle *alias);
pascal OSErr NewAliasMinimal(const FSSpec *target, AliasHandle *alias);
pascal OSErr NewAliasMinimalFromFullPath
                            (short fullPathLength,
                             const unsigned char *fullpath,
                             const Str32 zoneName, const Str31 serverName,
                             AliasHandle *alias);
pascal OSErr UpdateAlias    (const FSSpec *fromFile, const FSSpec *target,
                             AliasHandle alias, Boolean *wasChanged);
```

Resolving and Reading Alias Records

```
pascal OSErr ResolveAlias   (const FSSpec *fromFile, AliasHandle alias,
                             FSSpec *target, Boolean *wasChanged);
pascal OSErr MatchAlias     (const FSSpec *fromFile,
                             unsigned long rulesMask,
                             const AliasHandle alias, short *aliasCount,
                             FSSpecPtr aliasList, Boolean *needsUpdate,
                             AliasFilterProcPtr aliasFilter,
                             Ptr yourDataPtr);
pascal OSErr GetAliasInfo   (const AliasHandle alias, AliasInfoType index,
                             Str63 theString);
```

Application-Defined Routine

```
pascal Boolean MyMatchAliasFilter
                            (CInfoPBPtr cpbPtr, Boolean *quitFlag,
                             Ptr myDataPtr);
```

Assembly-Language Summary

Data Structure

Alias Record Data Structure

0	`userType`	long	file type of target file
4	`aliasSize`	word	size, in bytes, of record

Trap Macros

Trap Macro Requiring Routine Selectors

_AliasDispatch

Selector	Routine
$0002	NewAlias
$0003	ResolveAlias
$0005	MatchAlias
$0006	UpdateAlias
$0007	GetAliasInfo
$0008	NewAliasMinimal
$0009	NewAliasMinimalFromFullPath
$000C	ResolveAliasFile

Result Codes

noErr	0	No error
nsvErr	–35	The volume is not mounted
fnfErr	–43	Target not found, but volume and parent directory found
paramErr	–50	Parameter error
dirNFErr	–120	Parent directory not found
usrCanceledErr	–128	The user canceled the operation