Using the Alias Manager Creating and resolving an AliasRecord

You use the **Alias Manager** primarily to create and resolve an AliasRecord. You can also use it to get information about and update an AliasRecord.

The **Alias Manager** creates an AliasRecord 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 **DisposHandle** procedure, described in the **Memory Manager**. You can store and retrieve an AliasRecord 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 **File Manager** 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 a complete description of the **Gestalt** function, see the section entitled **Compatibility Guidelines** .

Creating Alias Records

You create a new AliasRecord by calling one of the three functions: **NewAlias** , **NewAliasMinimal** , or **NewAliasMinimalFromFullpath**. The **NewAlias** function creates a complete AliasRecord 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 provide a handle to it.

NewAlias 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 request that **NewAlias** store relative path information as well by supplying a starting point for a relative path (see **About Alias Records** for a description of relative path).

Call **NewAlias** when you want to create an AliasRecord 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 call **NewAlias** to create an AliasRecord for that file, including path information relative to the user's text file:

```
FSSpec *textFile;
FSSpec *target;
AliasHandle     myAliasHdl;

myErr= NewAlias (textFile, target, &myAliasHdl);
```

The *textFile* parameter is a pointer to a file system specification record that identifies the starting point for the relative search, in this case the user's text file. If you do not want relative path information recorded, pass a value of NIL in the first parameter.

The *target* parameter is a file system specification record that identifies the target file, in this example the dictionary file.

The *myAliasHdl* parameter is a variable in which the **Alias Manager** returns the handle to the AliasRecord that describes the target.

The two variations on the **NewAlias** function, **NewAliasMinimal** and **NewAliasMinimalFromFullpath** , record only a minimum of information about the target. **NewAliasMinimal** records only the target's name, parent directory ID, volume name and creation date, and volume mounting information. **NewAliasMinimalFromFullpath** 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 AliasRecord 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 AliasRecord for a target that doesn't exist or that resides on an unmounted volume.

Resolving Alias Records

The **Alias Manager** provides two alias-resolution functions:

- 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 **AliasRecord**, you call **ResolveAlias** . You call **MatchAlias** when you want your program to control the search.

This section describes the alias-resolution functions. The section entitled, **Search Strategies for Resolving Alias Records**, describes the underlying fast and exhaustive searches.

ResolveAlias

Typically, you call the **ResolveAlias** function to resolve an AliasRecord. **ResolveAlias** performs a fast search (described earlier in **Fast Search** under the section**Search Strategies for Resolving Alias Records**) and exits after it identifies one target.

By calling low-level functions, **ResolveAlias** compares some key information about the identified target with the information stored in the

AliasRecord. If any of the information is different, **ResolveAlias** automatically updates the record.

> **Note:** As with all other **Alias Manager** functions, **ResolveAlias** updates the record only in memory. Your application is responsible for updating an AliasRecord stored on disk when appropriate.

In the dictionary example illustrated in **About Alias Records**, the application calls **ResolveAlias** with a relative path specification when the user runs the spelling checker on a document with a customized dictionary.

myErr = **ResolveAlias** (textFile, myAliasHdl, &target, &wasChanged);

The *textFile* parameter is a pointer to a file system specification record that identifies the starting point for the relative search, in this case the user's text file. If you do not want relative path information used in the search, pass a value of NIL in the first parameter. If you provide a relative starting point, **ResolveAlias** performs the relative search first.

The *myAliasHdl* parameter is a handle to the AliasRecord to be resolved. In this example, the AliasRecord describes the dictionary file.

The *target* parameter is the file system specification record where the **Alias Manager** places the results of its search. After **ResolveAlias** completes, target contains the specification for the dictionary file.

The **ResolveAlias** function uses the wasChanged parameter to report whether it updated the AliasRecord. After **ResolveAlias** completes, wasChanged is TRUE if the record was updated and FALSE if it was not. If you are storing the AliasRecord, 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 AliasRecord, 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, use the FSSpec record to create a replacement for a missing file with the **File Manager** function **FSpCreate** .

MatchAlias

The **MatchAlias** function is a low-level routine that gives your application control over the searching algorithm.

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
- the order of the direct and relative searches in a fast search
- 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.

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 **MatchAlias** for a description of the filter function.

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

Maintaining Alias Records

You can store an AliasRecord as a resource of type 'alis'.

#define rAliasType 'alis'; resource type for saved an AliasRecord

To store and retrieve resources, use the standard **Resource Manager** functions (**AddResource** , **GetResource** , and **GetNamedResource**).

To update an AliasRecord, use the **UpdateAlias** function. You typically call **UpdateAlias** any time you know that the target of an AliasRecord 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 AliasRecord. It is the responsibility of your application to update the record.

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

Getting Information About Alias Records

To retrieve information from an AliasRecord 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.

Customizing Alias Records

An AliasRecord 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 AliasRecord for storing

additional data.

The **Alias Manager** stores the size of the record when it is created or updated in the aliasSize field. To customize an AliasRecord, you first increase the size of the record with the **SetHandleSize** procedure, described in the **Memory Manager**. 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 expand the record through the **Memory Manager** , the **Alias Manager** preserves your data, even if it changes the size of its own data when updating the record.

In general, you should customize only an AliasRecord that you have created.