NOTE: This Technical Note has been retired. Please see the Technical Notes page for current documentation.

Technical Note FL27
Mixing HFS and C File I/O

CONTENTS

This Technical Note discusses the problem of mixing calls to the Macintosh file system with calls to MPW C library file I/O routines.

[Aug 01 1989]

Problems with Communication Between HFS and C

Frequently, developers want to use both Macintosh file I/O and C file I/O. Developers who do this must keep in mind that they are combining two distinct file representations (the Macintosh and ANSI C). The only limitation on mixing HFS and C I/O functions is that they cannot be mixed on the same open file. There are three reasons why this cannot be done.

First, there is no routine that maps between a C `FILE` struct (returned by `fopen()`) to an HFS `fRefNum` (needed to call HFS functions). Similarly, there is no call to create a `FILE` struct given an `fRefNum` returned by `FSOpen()`. Thus, there is no way that the information from an `fopen()` call could be used to do a `fsread()`.

Second, even if the first problem were solved, the C libraries eventually call the HFS file system, but keep some internal state information. So, if you call HFS directly (say, `SetFPos()`), the C file system has no way of knowing a call was made and, therefore, doesn't update its state information.

Similarly, there is no mechanism for synchronizing the C library's buffers. For example, you perform an `fwrite()` with some number of characters which get put into a buffer without flushing it. Then you perform an `FSWrite()` with something else. Neither the C library nor HFS are aware that the other has written to the file.

Simply put, you cannot make HFS calls on a file opened with `fopen()` or `fdopen()`; you cannot use C library I/O on a file opened under HFS. However, here are some points to consider when manipulating the same file using both C and HFS. Keep in mind this isn't frequently done; there may be problems of which we are unaware.

One obvious problem is keeping track of the working directory. Be sure to save and restore the current working directory when moving between HFS and C I/O calls.

Following is an example routine, which mixes HFS and C I/O. Notice that it doesn't really solve the problem of mixing the two file systems, but rather it shows how to use `fopen()` with standard file (working directories or directory IDs) in general.

```
HardRockCocoJoe()
{
    Point            where;
    char            *prompt = "\pWe Are Here";
    char            *fname = "\pHardRockCocoJoe";
    FileFilterProcPtr    fileFilter = NULL;
    short            numTypes = 1;
    SFTypeList        typeList;
    DlgHookProcPtr    dlgHook = NULL;
    SFReply        reply;
    OSErr            result;
    FILE            *TheFile;
    short            fileNum;
    long            numofChars = 10;
    short            currentVRefNum;

    (void) GetVol(NULL, &currentVRefNum);

    result = FSOpen(fname, currentVRefNum, &fileNum);
    if (result != 0) {
        /* error checking */
    }
    else {
        result = FSWrite(fileNum, &numofChars, "from MacIO");
        if (result != 0) {
            /* error checking */
        }
    }

    (void) FSClose(fileNum);

    where.h = 80;
    where.v = 90;
    typeList[0] = 'TEXT';

    SFGetFile (where, prompt, fileFilter, numTypes, typeList, dlgHook, &reply);

    result = SetVol(reply.fName, reply.vRefNum /* from sfgetfile */);
    if (result != 0){
        /* error check */
    }

    p2cstr(reply.fName);
    TheFile = fopen (reply.fName, "a+");
    fprintf (TheFile, "\nfromC\n");
    fclose (TheFile);

    result = SetVol(NULL, currentVRefNum);
    if (result != 0){
        /* error check */
    }

    result = FSOpen(fname, currentVRefNum, &fileNum);
    if (result != 0){
        /* error check */
    }
    else {
        numofChars = 12;
        SetFPos(fileNum, fsFromLEOF, 0);
        result = FSWrite (fileNum, &numofChars, "from MacIO 2");
        if (result != 0) {
            /* error check */
        }
    }
}
```

Assuming the user chooses HardRockCocoJoe from the Standard File dialog box, the result of this routine is a file called HardRockCocoJoe, which contains the following data:

```
from MacIO
fromC
from MacIO 2
```

By keeping track of the working directory, you can work with HFS file I/O and C I/O. Of course, if you are working with many files, it could be a problem keeping track of the correct `paramBlock` and expensive to open and close the files each time you switch.

Another approach would be to construct a pathname from the Macintosh file system that could be passed to the C I/O functions. Technical Note #238, Getting a Full Pathname, goes into complete detail as to how this is done using either a working directory or `vRefNum` and `DirID`. But, this solution has serious drawbacks and is not recommended. One problem is that you have to manually create the pathname as a string and stuff the needed folder separators into that string. The current separator, the colon (:), may change in the future. A bigger problem is the length of the pathname. Currently, it can only be 256 characters, and that may be hard for you to guarantee. Lastly, there could be a problem if the user should change the directory or rename a file.

Back to top

You Were Warned

Be aware that you are responsible for any file problems you may have mixing HFS and C file I/O. If it can be avoided, by all means, avoid it.

Back to top

References

Inside Macintosh , Volume I, The Standard File Package

Inside Macintosh , Volume IV, The File Manager

Technical Note M.FL.FullPathName - Getting a Full Pathname

Back to top

Downloadables

	Acrobat version of this Note K)	Download

Back to top

Technical Notes by Date | Number | Technology | Title
Developer Documentation | Technical Q&As | Development Kits | Sample Code