Technical Note FL22

HFS Ruminations

CONTENTS

This technical note contains some thoughts concerning HFS.

[Jun 01 1986]

HFS numbers

A drive number is a small positive word (e.g., 3).

A `VRefNum` (as opposed to a `WDRefNum`) is a small negative word (e.g. `$FFFE`).

A `WDRefNum` is a large negative word (e.g. `$8033`).

A `DirID` is a long word (e.g. 38). The root directory of an HFS volume always has a `dirID` of 2.

Back to top

Working Directories

Normally an application doesn't need to open working directories (henceforth `WDs`) using `PBOpenWD`, since `SFGetFile` returns a `WDRefnum` if the selected file is in a directory on a hierarchical volume and you are running HFS. There are times, however, when opening a `WD` is desirable (see the discussion about `BootDrive` below).

If you do open a `WD`, it should be created with an `ioWDProcID` of `ERIK' (`$4552494B`) and it will be deallocated by the Finder. Note that under MultiFinder, ioWDProcID will be ignored, so you should only use `ERIK'.

`SFGetFile` also creates `WDs` with an `ioWDProcID` of `ERIK'. If `SFGetFile` opens two files from the same directory (during the same application), it will only create one working directory.

There are no `WDRefnums` that refer to the root--the root directory of a volume is always referred to by a `vRefNum`.

Back to top

When you can use HFS calls

All of the HFS `H' calls, except for `PBHSetVInfo`, can be made without regard to file system as long as you pass in a pointer to an HFS parameter block. `PBHGetVol`, `PBHSetVol` (see the warnings in the File Manager chapter of *Inside Macintosh*), `PBHOpen`, `PBHOpenRF`, `PBHCreate`, `PBHDelete`, `PBHGetFInfo`, `PBHSetFInfo`, `PBHSetFLock`, `PBHRstFLock` and `PBHRename` differ from their MFS counterparts only in that a `dirID` can be passed in at offset `$30`.

The only difference between, for example, `PBOpen` and `PBHOpen` is that bit 9 of the trap word is set, which tells HFS to use a larger parameter block. MFS ignores this bit, so it will use the smaller parameter block (not including the `dirID`). Remember that all of these calls will accept a `WDRefNum` in the `ioVRefNum` field of the parameter block.

`PBHGetVInfo` returns more information than `PBGetVInfo`, so, if you're counting on getting information that is returned in the HFS parameter block, but not in the MFS parameter block, you should check to see which file system is

active.

HFS-specific calls can only be made if HFS is active. These calls are: PBGetCatInfo, PBSetCatInfo, PBOpenWD, PBCloseWD, PBGetFCBInfo,PBGetWDInfo, PBCatMove and PBDirCreate. PBHSetVInfo has no MFS equivalent. If any of these calls are made when MFS is running, a system error will be generated. If PBCatMove or PBDirCreate are called for an MFS volume, the function will return the error code -123 (wrong volume type). If PBGetCatInfo or PBSetCatInfo are called on MFS volumes, it's just as if PBGetFInfo and PBSetFInfo were called.

Default volume

If HFS is running, a call to GetVol (before you've made any SetVol calls) will return the WDRefNum of your application's parent directory in the vRefNum parameter. If your application was launched by the user clicking on one or more documents, the WDRefNums of those documents' parent directories are available in the vRefNum field of the AppFile record returned from GetAppFiles.

If MFS is running, a call to GetVol (before you've made any SetVol calls) will return the vRefNum of the volume your application is on in the vRefNum parameter. If your application was launched by the user clicking on one or more documents, the vRefNum of those documents' volume are available in the vRefNum field of the AppFile record returned from GetAppFiles.

BootDrive

If your application or desk accessory needs to get the WDRefNum of the "blessed folder" of the boot drive (for example, you might want to store a configuration file there), it can not rely on the low-memory global BootDrive (a word at $210) to contain the correct value. If your application is the startup application, BootDrive will contain the WDRefNum of the directory/volume that your application is in (not the WDRefNum of the "blessed folder"); Your application could have been _Launched from an application that has modified BootDrive; if you are a desk accessory, you might find that some applications alter BootDrive.

To get the "real" WDRefNum of the "blessed folder" that contains the currently open System file, you should call SysEnvirons (discussed in Technical Note #129). If that is impossible, you can do something like this (Note: if you are running under MFS, BootDrive always contains the vRefNum of the volume on which the currently open System file is located):

```
...
CONST
    SysWDProcID = $4552494B; {"ERIK"}
    BootDrive = $210; {address of Low-Mem global BootDrive}
    FSFCBLen = $3F6; {address of Low-Mem global to
                      distinguish file systems }
    SysMap = $A58; {address of Low-Mem global that
                    contains system map reference number}

TYPE
    WordPtr = ^Integer; {Pointer to a word(2 bytes)}
...

FUNCTION HFSExists: BOOLEAN;

Begin {HFSExists}
    HFSExists := WordPtr(FSFCBLen)^ > 0;
End;  {HFSExists}


FUNCTION GetRealBootDrive: INTEGER;

VAR
     MyHPB        : HParamBlockRec;
     MyWDPB        : WDPBRec;
     err        : OSErr;
    sysVRef    : integer; {will be the vRefNum of open system's vol}

Begin {GetRealBootDrive}
    if HFSExists then Begin    {If we're running under HFS... }

        {get the VRefNum of the volume that }
        {contains the open System File      }
        err:= GetVRefNum(WordPtr(SysMap)^,sysVRef);

        with MyHPB do Begin
        {Get the "System" vRefNum and "Blessed" dirID}
            ioNamePtr    := NIL;
            ioVRefNum    := sysVRef; {from the GetVrefNum call}
            ioVolIndex    := 0;
```

```
            End; {with}
            err := PBHGetVInfo(@MyHPB, FALSE);

            with myWDPB do Begin      {Open a working directory there}
                ioNamePtr   := NIL;
                ioVRefNum   := sysVRef;
                ioWDProcID  := SysWDProcID; {Using the system proc ID}
                ioWDDirID  := myHPB.ioVFndrInfo[1];{ see Technote 67}
            End; {with}
            err := PBOpenWD(@myWDPB, FALSE);

            GetRealBootDrive := myWDPB.ioVRefNum;
            {We've got the real WD}
        End Else {we're running MFS}
            GetRealBootDrive := WordPtr(BootDrive)^;
            {BootDrive is valid under MFS}
    End;  {GetRealBootDrive}
```

From MPW C:

```
/*"ERIK"*/
#define    SysWDProcID    0x4552494B
#define    BootDrive    0x210
/*address of Low-Mem global that contains system map reference number*/
#define    SysMap    0xA58
#define     FSFCBLen     0x3F6
#define    HFSIsRunning ((*(short int *)(FSFCBLen)) > 0)

OSErr GetRealBootDrive(BDrive)
short int *BDrive;
{ /*GetRealBootDrive*/

    /*three different parameter blocks are used here for clarity*/
    HVolumeParam    myHPB;
    FCBPBRec        myFCBRec;
    WDPBRec        myWDPB;
    OSErr            err;
    short int        sysVRef; /*will be the vRefNum of open system's vol*/

    if (HFSIsRunning)

    { /*if we're running under HFS... */

    /*get the vRefNum of the volume that contains the open System File*/
        myFCBRec.ioNamePtr= nil;
        myFCBRec.ioVRefNum = 0;
        myFCBRec.ioRefNum = *(short int *)(SysMap);
        myFCBRec.ioFCBIndx = 0;

        err = PBGetFCBInfo(&myFCBRec,false);
        if (err != noErr) return(err);
    /*now we need the dirID of the "Blessed Folder" on this volume*/
        myHPB.ioNamePtr = nil;
        myHPB.ioVRefNum = myFCBRec.ioFCBVRefNum;
        myHPB.ioVolIndex = 0;

        err = PBHGetVInfo(&myHPB,false);
        if (err != noErr) return(err);

    /*we can now open a WD for the directory that
      contains the open system file one will most likely
      already be open, so PBOpenWD will just return that WDRefNum*/
        myWDPB.ioNamePtr = nil;
        myWDPB.ioVRefNum = myHPB.ioVRefNum;
        myWDPB.ioWDProcID = SysWDProcID; /*'ERIK'*/
            /* see Technote # 67 [c has 0-based arrays]*/
        myWDPB.ioWDDirID = myHPB.ioVFndrInfo[0];

        err = PBOpenWD(&myWDPB,false);
        if (err != noErr) return err;

        *BDrive = myWDPB.ioVRefNum; /*that's all!*/
```

```
    } /* if (HFSIsRunning) */
    else
        *BDrive = *(short int *)(BootDrive);
        /*BootDrive is valid under MFS*/

    return noErr;
} /*GetRealBootDrive*/
```

The Poor Man's Search Path (PMSP)

If HFS is running, the PMSP is used for any file system call that can return a file-not- found error, such as `PBOpen`, `PBClose`, `PBDelete`, `PBGetCatInfo`, etc. It is not used for indexed calls (that is, where `ioFDirIndex` is positive) or when a file is created (`PBCreate`) or when a file is being moved between directories (`PBCatMove`). The PMSP is also not used when a non-zero `dirID` is specified.

Here's a brief description of how the default PMSP works:

1. The directory that you specify (specified by `WDRefNum` or pathname) is searched; if the specified file is not found, then
2. the volume/directory specified by `BootDrive` (low-memory global at `$210`) is searched IF it is on the same volume as the directory you specified (see #1 above); if the specified file is not found, or the directory specified by `BootDrive` is not on the same volume as the directory that you specified, then
3. if there is a "blessed folder" on the same volume as the directory you specified (see #1 above), it is searched. Please note that if #2 above specifies the same directory as #3, then that directory is not searched twice. If no file is found, then
4. `fnfErr` is returned.

Back to top

ioDirId and ioFlNum

Two fields of the `HParamBlockRec` record share the same location. `ioDirID` and `ioFlNum` are both at offset `$30` from the start of the parameter block. This causes a problem, since, in some calls (e.g. `PBGetCatInfo`), a `dirID` is passed in and a file number is returned in the same field.

Future versions of Apple's HFS interfaces will omit the `ioFlNum` designator, so, if you need to get the file number of a file, it will be in the `ioDirID` of the parameter block after you have made the call. If you are making successive calls that depend on `ioDirID` being set correctly, you must "reset" the `ioDirID` field before each call. The program fragment in Technical Note #68 does this.

Back to top

PBHGetVInfo

Normally, `PBHGetVInfo` will be called specifying a `vRefNum`. There are times, however, when you may make the call and only specify a volume name. If this is so, there are a couple of things to look out for.

Let's say that we have two volumes mounted: "`Vol1:`" (the default volume) and "`Vol2:`". We also have a variable of type `HParamBlockRec` called `MyHPB`. We want to get information about `Vol2:`, so we put a pointer to a string (let's call it `fName`) in `MyHPB.ioNamePtr`. The string `fName` is equal to "`Vol2`" (Please note the missing colon). We also initialize `MyHPB.ioVRefNum` to 0. Then we make the call. We are very surprised to find out that we are returned an error of 0 (`noErr`) and that the `ioVRefNum` that we get back is not the `vRefNum` of `Vol2:`, but rather that of `Vol1:`.

Here's what's happening: `PBHGetVInfo` looks at the volume name, and sees that it is improper (it is missing a colon). So, being a forgiving sort of call, it goes on to look at the `ioVRefNum` field that you passed it (see pp. 99 of *Inside Macintosh*, vol. II). It sees a 0 there, so it returns information about the default volume.

If you want to get information about a volume, and you just have its name and you are not sure that the name is a proper one, you should set `MyHPB.ioVRefNum` to `-32768` (`$8000`). No `vRefNum` or `WDRefNum` can be equal to `$8000`. By doing this, you are forcing `PBHGetVInfo` to use the volume name and, if that name is invalid, to return a -35 error (`nsvErr`), "No such volume."

Back to top

PBGetWDInfo and Register D1

There was a problem with `PBGetWDInfo` that sometimes caused the call to inaccurately report the `dirID` of a directory. It is fixed in System 3.2 and later. To be absolutely sure that you won't get stung by this, clear register `D1` (`CLR.L D1`) before a call to `PBGetWDInfo`. You can do this either with an INLINE (Lisa Pascal and most C's) or with a short assembly-language routine before the call to `PBGetWDInfo`.

Back to top

References

The File Manager

Technical Note M.FL.ActiveFS -- Determining Which File System is Active

Technical Note M.FL.BlessedFolder -- Finding the "Blessed Folder"

Technical Note M.FL.SearchingVols -- Searching Volumes - Solutions and Problems

Back to top

Downloadables

 Acrobat version of this Note (56K) Download

Back to top
