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

Technical Note FL510

File Manager Directory Handling Q&As

CONTENTS

Implementing Macintosh System 7 desktop button

System 7 and copy-protect bit

How persistent must Macintosh directory IDs be?

Getting folder directory ID based on path name or parent folder

PBHDelete insights and code snippet

Working directories and WDRefNum

Getting HFS directories from full pathnames

How a new Macintosh directory ID is determined

Changing a file's fork without changing last-modified date

Code for detecting PBCatSearch support

PBHGetDirAccess ioNamePtr (not ioFileName) & ioDirID params

PBGetCatInfo NIL ioNamePtr bug and workaround

Forcing the Macintosh system to switch-launch

How can I tell which directory my Macintosh application is in?

Assembler code for distinguishing between HFS and MFS calls

Downloadables

This Technical Note contains a collection of archived Q&As relating to a specific topic--questions sent the Developer Support Center (DSC) along with answers from the DSC engineers. Current Q&As can be found on the Macintosh Technical Q&As web site.

[Oct 01 1990]

Back to top

Implementing Macintosh System 7 desktop button

Date Written: 6/22/92

Last reviewed: 6/14/93

We would like to know how to implement a desktop button under System 7 in a dialog other than the standard system Open dialog. Is there an easy way to find the mounted volumes and their icons and the files that are located on the desktop?

The items shown at the desktop level by Standard File are the mounted volumes and all items inside the invisible directory

named Desktop Folder at the root of the volume. You can find out what items are in the desktop folder by using PBGetCatInfo to enumerate the items in that directory. (For sample code showing the basic technique of indexing through the entries in a directory, see the Macintosh Technical Note "Searching All Directories on an HFS Volume.")

To get a list of volumes, you need to index through the list of volumes with PBHGetVInfo. The snippet code below shows how to do that. To get the icon associated with a specific volume, you need to make control call #21 to the volume's disk driver as documented on page V-470 of *Inside Macintosh* Volume V. The function GetDiskDriverInfo below shows the proper way to make the control call and safely return the icon and where string (the string that shows where a volume is in the Finder's Get Info dialog box).

```
FUNCTION GetDiskDriverInfo (driveNumber: Integer;
                                driverRefNum: Integer;
                                VAR iconHandle: Handle;
                                VAR whereString: Str255): OSErr;
VAR
    err: OSerr;
    pb: ParamBlockRec;
BEGIN
    iconHandle := NewHandle(kLargeIconSize);
    IF iconHandle = NIL THEN
        BEGIN    { bail out if memory couldn't be allocated }
            GetDiskDriverInfo := MemError;
            Exit(GetDiskDriverInfo);
        END;

    { get the disk driver info }
    WITH pb DO
        BEGIN
            ioVRefNum := driveNumber;
            ioRefNum := driverRefNum;
            csCode := 21;
        END;
    err := PBControlSync(@pb);
    IF err = noErr THEN
        BEGIN    { no error? then return the info }
            { move the icon into it }
            BlockMove(pb.ioMisc, iconHandle^, kLargeIconSize);

            { copy where information string into a string variable }
            whereString := StringPtr(ORD4(pb.ioMisc) + $100)^;
        END
    ELSE
        BEGIN    { error? then clean up and return no info }
            DisposHandle(iconHandle);
            iconHandle := NIL;
            whereString := '';
        END;
    GetDiskDriverInfo := err;
END;

Here's how you index through the list of volumes to get their names and
how to call the GetDiskDriverInfo function above to get the icon and
"where" string.

VAR
    hPB: HParamBlockRec;
    err: OSerr;
    dErr: OSErr;
    index: Integer;
    gVolName: Str255;
    gIconHandle: Handle;
    gWhereString: Str255;
```

```
BEGIN
    index := 1;
    REPEAT
        WITH hPB DO
            BEGIN
                ioNamePtr := @gVolName;
                ioVolIndex := index;
            END;
        err := PBHGetVInfoSync(@hPB);
        IF err = noErr THEN
            BEGIN
                dErr := GetDiskDriverInfo(hPB.ioVDrvInfo, hPB.ioVDRefNum,
                                          gIconHandle, gWhereString);
                IF dErr = noErr THEN
                    BEGIN
                { Do something with the volume name, icon, and where string. }
                { Don't forget to dispose the gIconHandle handle when you're }
                { done with the icon. }
                    END;
            END;
        index := index + 1;
    UNTIL err <> noErr;
END.
```

If you want to get fancy, you can use the custom icon attacted to some files (for example, alias files to AppleShare volumes). Chapter 9 of *Inside Macintosh* Volume VI talks about custom icons and alias files on pages 9-28 through 9-32. You can tell if a file or directory has a custom icon by looking at the `hasCustomIcon` bit in the `fdFlags` word of the `FInfo` record returned by `PBGetCatInfo` (which you call to build the list of files and folders).

Back to top

System 7 and copy-protect bit

Date Written: 4/26/90

Last reviewed: 6/14/93

I need to prevent users from copying my application off a volume. Is there a new equivalent of the old Bozo bit?

The Bozo or `NoCopy` bit was bit 11 in the `fdFlags` word of the `FInfo` record. As noted in the Macintosh Technical Note "Finder Flags," this bit hasn't been used for that purpose since System 5. In fact, System 7 reused bit 11 for the `isStationery` bit. (See *Inside Macintosh* Volume VI, pages 9-36 and 9-37, for the current list of Finder flag bits.)

There isn't an equivalent of the Bozo bit. However, the System 7 Finder won't copy files that have the copy-protect bit (bit 6) set in the `ioFlAttrib` field returned by the `PBGetCatInfo` function. However, the bits in the `ioFlAttrib` field can't be changed with the `PBSetCatInfo` function. Instead, they're used to report the state of things set by other parts of the file system.

The copy-protect bit is set by the AppleShare external file system when it finds that a file's copy-protect bit is returned by an AppleTalk Filing Protocol file server. The AppleShare external file system is the only file system we know of that sets the copy-protect bit. There's no way to make the local file system set the copy-protect bit for volumes it controls.

Back to top

How persistent must Macintosh directory IDs be?

Date Date Written: 8/28/92

Last reviewed: 11/1/92

How persistent must directory IDs be? In our external file system, directory IDs are generated starting from

fsRtDirID whenever a volume is mounted. When one of our volumes is unmounted and then remounted without rebooting the Macintosh in between, the Finder immediately does a GetCatInfo specifying a directory ID (5, I think it's looking for the Trash folder). This would imply that directory IDs need to persist across dismounts. Is this the case? Is there any way to get the Finder to forget the directory IDs it has stored, besides rebooting?

The Macintosh file system only expects directory ID numbers to be fixed while the volume is mounted.

However, other parts of the Macintosh operating system (for example, the Alias Manager and the Finder) and applications expect directory IDs to persist between volume mounts so that they can find a particular directory when a volume is remounted. For example, under System 6, the correct way to store the location of a file is by saving the volume name, the volume's creation date (to help find the volume in the case where it is renamed), and the parent directory ID number. The Alias Manager under System 7 (and under System 6 when QuickTime is installed) uses the same technique when it creates an alias to a file or directory.

The Finder keeps the directory ID numbers of the directories it can find with the FindFolder function in the volume's FndrInfo record, so that's why you're seeing the Finder look for a particular directory ID.

Back to top

Getting folder directory ID based on path name or parent folder

Date Written: 4/26/90

Last reviewed: 6/14/93

Is there any Macintosh routine that gets the directory ID of a folder based on

the path name? How about if I know the parent folder's directory ID?

You can call PBGetCatInfo with ioFDirIndex set to zero to get the directory ID of a folder based on the path name. Call PBHGetVInfo to find its vRefNum.

You can also call PBGetCatInfo to get a folder's dirID based on its name and its parent's dirID and vRefNum. Here's an example:

```
{ given a folder's name, vRefNum, and parent dirID, find its dirID }
FUNCTION GetFolderDirID (parentFolderVRefNum: Integer;
                         parentFolderDirID: LongInt; folderName: Str63;
                         VAR folderDirID: LongInt): OSErr;
  VAR
    myCInfoPBRec: CInfoPBRec;
    retCode: OSErr;
  BEGIN
    myCInfoPBRec.ioCompletion := NIL;
    myCInfoPBRec.ioNamePtr := @folderName;
    myCinfoPBRec.ioVRefNum := parentFolderVRefNum;
    myCInfoPBRec.ioFDirIndex := 0; { use name, vRefNum, dirID }
    myCInfoPBRec.ioDrDirID := parentFolderDirID; { will be changed }

    retCode := PBGetCatInfoSync(@myCInfoPBRec); { IM IV-155 }

    IF retCode = noErr THEN folderDirID := myCInfoPBRec.ioDrDirID;

    GetFolderDirID := retCode;
  END;
```

Back to top

PBHDelete insights and code snippet

Date Written: 11/28/90

Last reviewed: 2/8/91

Using a synchronous call to delete an empty Macintosh directory using PBHDelete returns a fBsyErr.

I don't know exactly what is causing the fBsyErr on your machine, but here's my best guess: The reason you are experiencing this problem is that something has set a working directory to the one you're trying to delete. I was able to get this to happen by setting the current directory in MPW to "foo:temp:" and then trying to delete it. The bad news is that you can't close another process's working directory. There will be cases where your application won't be able to delete the directory until the user resets the default directory in the offending application. Such are the perils of working in a multitasking environment....

Also, I would suggest using a routine like the one included below. It ensures that the specified file is in fact a directory, and supplies a vRefNum and DirID to PBHDelete, which I feel is safer than just giving a full pathname.

```
/*
**
**        DirDelete.c
**
**            An MPW tool to delete a directory.
**            Gets the target file from the command line.
**
**            Neil Day
**            MacDTS
**            Apple Computer, Inc.
**            11/29/90
**
*/

#include     <Stdio.h> /* compiler interfaces */
#include     <Files.h>
#include     <Memory.h>
#include    <Strings.h>

#define        DIR_BIT        4

OSErr DirDelete (char *dirName); /* function prototypes */

main (argc,argv)
int argc;
char **argv;
{
    int argCounter;
    OSErr err;

    if (argc < 2) {
        printf ("Format :  DirDelete <file1> [<file2>
        ...  <fileN>]\n");
        return;
    }

    for (argCounter = 1; argCounter < argc; argCounter++) {
        printf ("%s\n",argv[argCounter]);
        err = DirDelete (argv[argCounter]);
        if (err)
            printf ("DirDelete :  got error %d while trying to
            remove %s\n",

                        err,argv[argCounter]);
    }
```

```
}

/*
**   DirDelete
**     Deletes a directory given the full pathname to the directory.
**
**     Uses PBGetCatInfo to retrieve the vRefNum and DirID of the directory
**     and its status flags.  Checks to see that the specified file is
**     in fact a directory, and deletes it if it is...
**
*/
OSErr DirDelete (dirName)
char *dirName;
{
    OSErr err;
    CInfoPBPtr Info;
    HParmBlkPtr delete_me;

    err = noErr;

    Info = (CInfoPBPtr) NewPtrClear (sizeof(CInfoPBRec));
    Info->hFileInfo.ioNamePtr = c2pstr(dirName);

    delete_me = (HParmBlkPtr) NewPtrClear (sizeof (HParamBlockRec));

    err = PBGetCatInfo (Info,false);

    if ((!err) && ((Info->hFileInfo.ioFlAttrib>>DIR_BIT)
        & 0x01)) {
        delete_me->fileParam.ioVRefNum = Info->hFileInfo.ioVRefNum;
        delete_me->fileParam.ioDirID = Info->hFileInfo.ioDirID;
        err = PBHDelete (delete_me,false);
    }

    DisposPtr ((Ptr)Info);
    DisposPtr ((Ptr)delete_me);
    return err;
```

Back to top

Working directories and WDRefNum

Date Written: 12/13/90

Last reviewed: 6/14/93

How do I get the directory ID when using `HCreateResFile` (as described in the Macintosh Technical Note "New Resource Manager Calls." Also, what exactly are working directories, and do I need to be concerned with them, or are they a thing of the past?

The directory ID comes indirectly from the Standard File dialogue that the user uses to select an output or input file. Standard File returns a working directory reference number in the `ioVRefNum`. Use the `PBGetWDInfo` call to break that up into an `ioVRefNum` and `ioDirID` pair.

Working directories are a compatibility item to allow programs written for MFS to run under HFS. Since MFS didn't support directories in any true sense, HFS needed to be able to encode both volume and directory information in the same place. The vRefNum was the most logical spot to do that, and hence the `WDRefNum` was born. The first and last thing you should do with a `WDRefNum` is convert it to a `vRefNum` and `DirID` pair. For more information on the `PBGetWDInfo` call, please refer to *Inside Macintosh* Volume IV, page 159.

X-Ref:

DTS Macintosh Technote "New Resource Manager Calls"

Inside Macintosh Volume IV, File Manager chapter, page 159

Back to top

Getting HFS directories from full pathnames

Date Written: 12/21/90

Last reviewed: 1/16/91

How can I determine Macintosh HFS directories from the full pathnames without going through SFGetFile or opening files?

———

Apple recommends not using full pathnames. They're unwieldy, they take up lots of space, the File Manager doesn't deal with names longer than 255 characters, and they're unnecessary.

However, if you are working with a piece of software other than your own, it may be unavoidable. If so, there are two methods depending on the size of the full pathname:

1. If the full pathname is less than 255 characters, simply get the dirID of the directory by calling PBGetCatInfo.

```
    VAR
            pbc:          CInfoPBRec;

    BEGIN
            name := 'Chinese Junk:Test:VMCalls:';

            WITH pbc DO BEGIN
                ioCompletion := NIL;
                ioNamePtr := @name;
                ioVRefNum := 0;
                ioFDirIndex := 0;
                ioDrDirID := 0;
            END;
            err := PBGetCatInfo(@pbc, FALSE);
```

This will return to you the dirID of the folder in ioDrDirID.

2. If the full pathname is longer than 255 characters, you will have to break up the name into smaller parts and get information incrementally. This is shown below. Note that this example is not quite realistic since I store the original full pathname in a Pascal string, which will always be smaller than 255 characters.

```
VAR
        pbc:                CInfoPBRec;
        pbh:                HParamBlockRec;
        name:               Str255;
        tempname:           Str255;
        vRefNum:            Integer;
        dirID:              LONGINT;
        done:               Boolean;
        lastPos:            Integer;
        curPos:             Integer;

BEGIN
        name := 'Chinese Junk:Test:VMCalls:';

        tempname := name; {PBHGetVInfo modified ioNamePtr string,
```

```
        so copy it}
        WITH pbh DO BEGIN
            ioCompletion := NIL;
            ioNamePtr := @tempname;
            ioVRefNum := 0;
            ioVolIndex := 0;
        END;
        err := PBHGetVInfo(@pbh, FALSE);

        vRefNum := pbh.ioVRefNum; {get vRefNum for PBGetCatInfo calls}
        dirID := 0; {init dirID to zero}
        done := FALSE;
        lastPos := 1;

        REPEAT
            curPos := lastPos; {look for the next chunk surrounded
            by colons}
            REPEAT
                curPos := curPos + 1;
                done := curPos >= LENGTH(name);
            UNTIL (name[curPos] = ':') | (done);
            tempname := Copy(name, lastPos, curPos - lastPos + 1);
            lastPos := curPos;
            WITH pbc DO BEGIN
                ioCompletion := NIL;
                ioNamePtr := @tempname;
                ioVRefNum := vRefNum;
                ioFDirIndex := 0;
                ioDirID := dirID;
            END;
            err := PBGetCatInfo(@pbc, FALSE);
            dirID := pbc.ioDrDirID; {get this dirID to use a parent
                                     for next chunk}

            Writeln('err = ', err);
            Writeln('tempName = ', tempname);
            Writeln('ioVRefNum = ', pbc.ioVRefNum);
            Writeln('ioDrDirID = ', pbc.ioDrDirID);
            Writeln('ioDrParID = ', pbc.ioDrParID);

        UNTIL done;
END.

This will give you the following output:

    err =        0
    tempName = Chinese Junk:
    ioVRefNum =       -2
    ioDrDirID =        2
    ioDrParID =        2
    err =        0
    tempName = :Test:
    ioVRefNum =       -2
    ioDrDirID =      271
    ioDrParID =        2
    err =        0
    tempName = :VMCalls:
    ioVRefNum =       -2     <--- this is the vRefNum you want
    ioDrDirID =    26399     <--- this is the dirID you want
    ioDrParID =      271
```

Back to top

How a new Macintosh directory ID is determined

Date Written: 1/24/91

Last reviewed: 2/14/91

Our Macintosh application moves files between directories on a file system; the directories determine certain actions and are seldom changed; thus, the application keeps a persistent reference to the directories. The current form of this reference is <volume name, path from root>. According to a System 7 lecture, the preferred form for a persistent file reference is <volume name, directory ID, filename>, which reduces to <volume name, directory ID>, the preferred form for a persistent directory reference. The process of throwing folders away and creating new ones could result in a directory ID being reused on an HFS volume. Is this truly a threat, or has the directory allocation scheme been designed to require many transactions before reusing an ID?

———

You are correct in your assumption that many transactions need to take place before a directory ID gets reused. The system does not start with dir ID = 0 and search the hard disk until it finds an ID that's not used. Instead, the ID of the last created folder is in a "last created ID" field on the disk. When the system creates a new folder, it searches with that "last value" + 1. Since these are 32-bit values, you'd have to create over 4 billion folders before you wrap around again to zero.

Back to top

Changing a file's fork without changing last-modified date

Date Written: 7/9/91

Last reviewed: 6/14/93

Inside Macintosh Volume VI, page 2-22, recommends updating the window positions in a file without changing the last modification date and time on the file. How do I alter a file without automatically changing the timestamp?

———

To modify the contents of a file's data or resource fork without changing the last modified date, get the modified date before performing any save operations on the file and restore it when you're done. You can use the `PBHGetFInfo` and `PBHSetFInfo` calls to do this. A short Pascal snippet that modifies the contents of a known file's resource fork without modifying its modification date is shown below. The code shows how the parameter block is filled in with the file's information at the start of the routine with a `PBHGetFInfo` call, and the same data is then used without modification to set the file information at the end of the routine with a `PBHSetFInfo`. *Inside Macintosh* Volume IV, page 150, tells you which fields can be changed with `PBHSetFInfo`.

```
Procedure DummyResource;

var    a:stringHandle;
    b:Integer;
    fred:str255;
    theBlock:HParamBlockRec;
    anErr:OSErr;

begin
  {Set up the parameter block}
  fred:='anIcon';
  theBlock.ioCompletion:=nil;
  theBlock.ioNamePtr:=@Fred;
  theBlock.ioVRefNum:=0;
  theBlock.ioFDirIndex:=0;
  theBlock.ioDirID:=0;
 {Recover the files info to save the mod date}
  anErr:=PBHGetFInfo(@theBlock,false);
 {modify the resource fork}
  b:=OpenResFile('anIcon');
  a:=StringHandle(GetResource('STR ',128));
  a^^[1]:=char(ord(a^^[1])+1);
  ChangedResource(Handle(a));
  UpdateResFile(b);
  CloseResFile(b);
 {Now restore the original last mod date in the files directory entry.}
  theBlock.ioCompletion:=nil;
  theBlock.ioNamePtr:=@Fred;
  theBlock.ioVRefNum:=0;
  theBlock.ioFDirIndex:=0;
  theBlock.ioDirID:=0;
  anErr:=PBHSetFInfo(@theBlock,false);
end;
```

Back to top

Code for detecting PBCatSearch support

Date Written: 9/26/91

Last reviewed: 10/15/91

Our code for volume searching using PBCatSearch doesn't work for AppleShare volumes. Do you know how we can make it work?

———

PBCatSearch works only on volumes that support it. If you attempt to use PBCatSearch on a volume that doesn't support it, you'll get a wrgVolTypErr (-123). (This isn't documented in *Inside Macintosh* Volume VI.)

AppleShare 2.0 volumes (AppleShare 2.0 is an AFP 2.0 server) and volumes supported by some other external file systems do not support PBCatSearch, so you'll have to resort to recursively searching the catalog using a routine similar to the one shown in the Macintosh Technical Note "Searching Volumes--Solutions and Problems" on those volumes. PBCatSearch is supported by AppleShare volumes on System 7 File Sharing servers because File Sharing supports AFP version 2.1 which includes CatSearch as a supported command.

You can use the File Manager call PBHGetVolParms to check a volume for most volume-specific features, including support for PBCatSearch. Here's a short function that shows how to do that:

```
FUNCTION CatSearchable (vRefNum: Integer): Boolean;
{See if PBCatSearch is supported on the volume specified by vRefNum}
  VAR
    pb: HParamBlockRec;
    infoBuffer: GetVolParmsInfoBuffer;
    err: OSErr;
BEGIN
  WITH pb DO
    BEGIN
      ioNamePtr := NIL;
      ioVRefNum := vRefNum;
      ioBuffer := @infoBuffer;
      ioReqCount := SizeOf(infoBuffer);
    END;
  err := PBHGetVolParms(@pb, FALSE);
  IF err = noErr THEN
    IF BTst(infoBuffer.vMAttrib, bHasCatSearch) THEN
      CatSearchable := TRUE
    ELSE
      CatSearchable := FALSE
  ELSE
    CatSearchable := FALSE;
END;
```

Back to top

PBHGetDirAccess ioNamePtr (not ioFileName) & ioDirID params

Date Written: 12/20/91

Last reviewed: 6/14/93

Two `PBHGetDirAccess` parameters listed in *Inside Macintosh* Volume V, page V-394, `ioFileName` and `ioDirID`, aren't in the MPW C 3.2 header files. Can anyone show me the correct way to call this function?

———

The ioFileName parameter is really `ioNamePtr` (*Inside Macintosh* Volume V documentation error) and can be found in the `ParamBlockHeader`. The ioDirID parameter is in the `FileParam` variant of the `ParamBlockRec`, so if you just refer to it as `myHParamBlockRec.ioDirID`, you should be fine. You'll find similar cases in the C include files where you'll need to use two different variants to get to all the fields used by a call.

Back to top

PBGetCatInfo NIL ioNamePtr bug and workaround

Date Written: 3/6/92

Last reviewed: 6/14/93

I'm using `PBGetCatInfo` to index through the entries in a directory. Because I'm not interested in the names of the directory entries, I set `ioNamePtr` to NIL. Everything worked great until I tried this on a volume shared with Macintosh File Sharing. After running my program on a shared volume, the system started acting unpredictably. What happened?

———

The problem you reported with `PBGetCatInfo` is caused by a bug in the Macintosh File Sharing code. It doesn't check for a NIL value in `ioNamePtr` correctly and *always* copies the file or folder name found at a particular catalog position to whatever `ioNamePtr` points to. This means up to 32 bytes starting at location 0 will be trashed if `ioNamePtr` = NIL. So until this problem is fixed, you should avoid using indexed calls to `PBGetCatInfo` when `ioNamePtr` = NIL; `ioNamePtr` should always point to storage for a Str255.

The AppleShare 3.0 file server does not have this problem, nor does System 7.1 Macintosh File Sharing.

Back to top

Forcing the Macintosh system to switch-launch

Date Written: 11/21/89

Last reviewed: 11/21/90

How can I get my Macintosh application to force the system to switch to the application's disk when the user launches it, like Installer does?

———

This is called switch-launching. You can do what Installer does, setting a Finder flag that tells Finder to switch-launch to the new volume. Set this flag with ResEdit by selecting the application, selecting Get Info from the File menu, then setting the appropriate checkbox. You can also set this with the MPW command:

```
SetFile -a A foo
```

Note that forcing a switch-launch in this way *never* works if any of the following conditions are true:

* The new volume is not bootable--that is, has no system or Finder or is a volume like an AppleShare volume;

* MultiFinder is running;

* Either the system or Finder on the new volume is an older version than the one currently in use.

If any of these conditions are true and you try to force a switch-launch, the user will get a dialog stating why switch-launching is not possible, and asking if the user wants to proceed with the launch (without the switch) or cancel it.

Because of these limitations, if you plan for your application to be used by others and you depend on this capability, you need to think carefully about the consequences of forcing switch-launching. For example, a user may (and probably will) choose to run MultiFinder. There is no way for you to know if MultiFinder is running, and if you are depending on the switch, strange things may happen. Switch-launching may be even more restricted in the future.

X-Ref:

Macintosh Technical Note "Finder Flags"

Back to top

How can I tell which directory my Macintosh application is in?

Date Written: 5/3/89

Last reviewed: 11/21/90

How can I tell which directory my Macintosh application is in?

———

When an application is started, the default volume is set to the directory that contains the application. `GetVol` returns the default volume. If the application calls `GetVol` before changing the default volume, it will have the directory for the folder containing the application.

Back to top

Assembler code for distinguishing between HFS and MFS calls

Date Written: 3/9/92

Last reviewed: 6/14/93

I'm trying to patch the Macintosh `_Create` trap and save off the volume and directory ID of the file that is being created.

How do I tell if the caller has placed a HParamBlkPtr or a ParamBlkPtr in A0 so I'll know whether to treat the ioDirID element as significant?

When you're called from the trap dispatcher, the trap word you're being called with is placed in register D1. Since HFS calls can be distinguished from MFS calls by bit 9 being set in the trap word, you can simply see if this bit is set in D1 to determine whether the call is using an HParamBlkPtr. Here's an example in 68000 of how you could check this bit:

```
createPatch movem.l a0/d0-d1,-(sp) ; save registers
            move.l  d1,d0
            andi.w  #$0200,d0
            beq.s   isCreate
isHCreate   ...
            ...
            bra.s   done
isCreate    ...
            ...
done        movem.l  (sp)+,a0/d0-d1
            move.l  createLink,-(sp)
            rts
```

Besides checking HFS vs. MFS, you can also use D1 to determine whether or not the call was made asynchronously by checking bit 10 of D1.

Back to top

Downloadables

Acrobat version of this Note (K) Download

Back to top
