Which File System Is Active Different volumes may have different File Systems

Under certain circumstances it is necessary to determine which file system is currently running on a particular volume. For example, on a 64K ROM machine, some applications may need to check for MFS vs HFS. If your application only runs on 128K ROMs or newer, you do not need to check for HFS, since it exists on all machines except those with the original 64K ROMs. Still, you may need to check if a particular volume is in High Sierra, ISO 9660 or audio CD format.

Before performing these file system checks, be sure to call **SysEnvirons** to make sure the machine has ROMs that know about the calls you need.

To check for HFS on 64K ROM machines, check the low-memory global FSFCBLen. This global is -1 if MFS is active and a positive number (currently 0x5E) if HFS is active. Even if the application determines that it is running under HFS, it should not assume that all mounted volumes are HFS. To check individual volumes for HFS, call **PBHGetVInfo** and check the directory signature (the ioVSigWord field of an HParamBlockRec). A directory signature of 0xD2D7 means the volume is an MFS volume, while a directory signature of 0x4244 means the volume is an HFS volume. The following example can be used to get information about all mounted volumes.

Example

```
OSErr GetIndVolume (short whichVol, char *volName, short *volRefNum)
{
    /* Return the name and vRefNum of volume specified by whichVol */

    HVolumeParam    volPB;
    OSErr           error;

    volPB.ioNamePtr = volName;          /* make sure it returns the name */
    volPB.ioVRefNum = 0;                /* 0 means use ioVolIndex */
    volPB.ioVolIndex = whichVol;        /* use this to determine volume */

    error = PBHGetVInfo(&volPB,false); /* do it */
    if(error == noErr)
        *volRefNum = volPB.ioVRefNum;   /* return the volume reference */

    /* other information is available from this record; see the File Manager*/
    /* description of PBHGetVInfo for more details... */

    return (error);

}
```

This routine can be called several times to get information about all mounted volumes, starting with whichVol = 1, and incrementing whichVol until the routine returns nsvErr (-35, "no such volume").

Knowing which folder on an HFS disk contains the System File and Finder can be very useful information. This is available from **SysEnvirons**. Sometimes

it can also be useful to find the "blessed" folder (the folder that contains both the System File and the Finder) on HFS volumes other than the one containing the currently open System File. The following example is similar to the one given above, but adds an extra call to see if each mounted volume contains a "blessed" folder.

Example

```
#include <Files.h>

OSErr GetBlessed (short vRefNum, long *blessed)
{
    /* determine if volume contains a "blessed" folder */

    HParamBlockRec  myHBP;
    OSErr  error;

    *blessed = 0L;

    myHBP.volumeParam.ioNamePtr = 0L;
    myHBP.volumeParam.ioVRefNum = vRefNum;    /* get for default volume */
    myHBP.volumeParam.ioVolIndex = 0;            /* not making indexed calls */
    error = PBHGetVInfo(&myHBP, false);
    if(error == noErr)
        blessed = myHBP.volumeParam.ioVFndrInfo;

    return error;
}

main()
{
    short whichVol = 1;
    char *volName;
    HVolumeParam volPB;
    OSErr error = 0;
    long     dirID;

    volName = (char *)NewPtr(255);

    while(error != nsvErr)
    {
        volPB.ioNamePtr = volName;
        volPB.ioVRefNum = 0;
        volPB.ioVolIndex = whichVol++;

        error = PBHGetVInfo(&volPB,false);

        if(error == noErr)
            GetBlessed(volPB.ioVRefNum,&dirID);
    }
}
```

The dirID of the blessed folder is myHPB.ioVFndrInfo[0]. If it is 0, it means

that there is no blessed folder (or it is an MFS volume). If it is 2 (fsRtDirID), it means that the System File and Finder are at the root level. myHPB.ioVFndrInfo[1] is the dirID of the startup application. Note that these calls will not work on machines without HFS, since they have no **PBHGetVInfo** call, and the ioVFndrInfo field does not exist in the **PBGetVInfo** call.