**PBGetCatInfo** Query file or directory date/time, attributes, etc.

#include <Files.h> **File Manager (PBxxx)**

| | | |
|---|---|---|
| OSErr | **PBGetCatInfo(***pb*, *async* **)**; | |
| CInfoPBPtr | *pb* ; | address of a 108-byte CInfoPBRec union |
| Boolean | *async* ; | 0=await completion; 1=immediate return |
| | ***returns*** | Error Code; 0=no error |

**PBGetCatInfo** obtains information describing both files and directories. It is most useful for indexing through all items in a directory. You can use the **PBGetCatInfo** function to determine whether a file has a file ID. The value of the file ID is returned in the ioDirID field. Because that parameter could also represent a directory ID, call **PBResolveFileIDRef** to see if the value is a real file ID. If you want to both determine whether a file ID exists for a file and create one if it doesn't, use **PBCreateFileIDRef**, which either creates a file ID or returns fidExists

*pb* is the address of a 108-byte CInfoPBRec union. The layout of the returned information is either a 108-byte HFileInfo or a 104-byte DirInfo structure, depending upon whether the item queried is a file or a directory.

To obtain and interpret file information, use an HFileInfo structure:

| Out-In | Name | Type | Size | Offset | Description |
|---|---|---|---|---|---|
| –> | ioCompletion | ProcPtr | 4 | 12 | Completion routine address (if *async* =TRUE) |
| <- | ioResult | OSErr | 2 | 16 | Error Code (0=no error, 1=not done yet) |
| <-> | ioNamePtr | StringPtr | 4 | 18 | Entry: Address of full or partial path/filename<br>Return: Receives 32-byte max file name |
| –> | ioVRefNum | short | 2 | 22 | Volume, drive, or working directory reference |
| <- | ioFRefNum | short | 2 | 24 | File reference number |
| –> | ioFDirIndex | short | 2 | 28 | Index (0=query 1 itm,-1=query dir in ioDrDirID) |
| <- | ioFlAttrib | SignedByte | 1 | 30 | File Attribute bits (bit 0=locked, bit 4=dir, etc.) |
| <- | ioFlFndrInfo | FInfo | 16 | 32 | (File type, creator, flags, icon point, etc.) |
| <-> | ioDirID | long | 4 | 48 | Entry: "hard" ID of directory to search<br>Return: "hard" file ID or directory ID |
| <- | ioFlStBlk | short | 2 | 52 | First allocation block of data fork |
| <- | ioFlLgLen | long | 4 | 54 | Logical end-of-file of data fork |
| <- | ioFlPyLen | long | 4 | 68 | Physical end-of-file of data fork |
| <- | ioFlRStBlk | short | 2 | 62 | First allocation block of resource fork |
| <- | ioFlRLgLen | long | 4 | 64 | Logical end-of-file of resource fork |
| <- | ioFlRPyLen | long | 4 | 68 | Physical end-of-file of resource fork |
| <- | ioFlCrDat | long | 4 | 72 | Date/Time of creation |
| <- | ioFlMdDat | long | 4 | 76 | Date/Time of last modification |
| <- | ioFlBkDat | long | 4 | 80 | Date/Time last backed up |
| <- | ioFlXFndrInfo | FXInfo | 16 | 84 | (Icon ID, comment, home directory) |
| <- | ioFlParID | long | 4 | 100 | "hard" ID of dir in which this file resides |
| <- | ioFlClpSiz | long | 4 | 104 | Clump size for optimal writing (0=vol default) |

To obtain and interpret directory information, use a DirInfo structure:

| Out-In | Name | Type | Size | Offset | Description |
|---|---|---|---|---|---|
| –> | ioCompletion | ProcPtr | 4 | 12 | Completion routine address (if *async* =TRUE) |
| <- | ioResult | OSErr | 2 | 16 | Error Code (0=no error, 1=not done yet) |
| <-> | ioNamePtr | StringPtr | 4 | 18 | Entry:Address of full or partial path/filename<br>Return: Receives 32-byte max directory name |
| –> | ioVRefNum | short | 2 | 22 | Volume, drive, or working directory reference |
| <- | ioFRefNum | short | 2 | 24 | File reference number |
| –> | ioFDirIndex | short | 2 | 28 | Index (0=query 1 itm,-1=query dir in ioDrDirID) |

| | | | | | |
|---|---|---|---|---|---|
| <- | ioFlAttrib | SignedByte | 1 | 30 | File Attribute bits (locked, directory, etc) |
| <- | ioDrUsrWds | DInfo | 16 | 32 | (Folder rectangle, location, flags, etc.) |
| <-> | ioDrDirID | long | 4 | 48 | Entry: "hard" ID of directory to search<br>Return: ID of directory found (if item is a dir) |
| <- | ioDrNmFls | short | 2 | 52 | Number of files and directories in this directory |
| <- | ioDrCrDat | long | 4 | 72 | Date/Time of creation |
| <- | ioDrMdDat | long | 2 | 76 | Date/Time of last modification |
| <- | ioDrBkDat | long | 4 | 80 | Date/Time last backed up |
| <- | ioDrFndrInfo | DXInfo | 16 | 84 | (Scroll point, home dir, comment, etc.) |
| <- | ioDrParID | long | 4 | 100 | "hard" ID of this directory's parent |

*async* is a Boolean value. Use FALSE for normal (synchronous) operation or TRUE to enqueue the request and resume control immediately. See Async I/O.

**Returns**: an operating system Error Code. It will be one of:

| | | |
|---|---|---|
| noErr | (0) | No error |
| bdNamErr | (-37) | Bad name |
| dirNFErr | (-120) | Directory not found |
| extFSErr | (-58) | External file system |
| fnfErr | (-43) | File not found |
| ioErr | (-36) | I/O error |
| nsvErr | (-35) | No such volume |
| paramErr | (-50) | No default volume |
| volGoneErr | (-124) | Server volume disconnected |
| accessDenied | (-5000) | User access privileges incorrect |
| denyConflict | (-5006) | Conflict between user deny mode and volume setting |
| noMoreLocks | (-5015) | Sever can't lock any more ranges |
| rangeNotLocked | (-5020) | User attempt to unlock somebody else's range |
| rangeOverlap | (-5021) | User attempt to lock previously locked range |

Notes: For getting information about files, use **PBGetCatInfo** in the same way as **PBHGetFInfo**. One advantage of **PBGetCatInfo** is that its parameter block contains an FXInfo structure, identifying the file's icon, a comment, and its home directory.

To get information about a single specific item set ioNamePtr to the Pascal-style name of the file (optionally prefixed with a multiple-name path) and set ioVRefNum to identify the volume or working directory. Set ioDirID instead of of ioVRefNum to search a "hard" directory ID.

The ioACuser field returns information on the access rights to server volume directories, such information being in the form of a SignedByte where: bit 7 is set if the user is not the owner of the directory: bits 6 through 3 are reserved and set to 0: bit 2 is set if the user is not allowed to Make Changes: bit 1 is set if the user does not have Read privileges; and bit 0 is set if the user does not have Search privileges. A return of zero, therefore, means that the user owns and has complete privileges to the directory. The **AccessParam** data structure description includes a diagram of the format.

When indexing (scanning through entries), the ioNamePtr field must point to a 32-byte minimum buffer to receive the file name (or use NIL, if you don't care about the name). Again, identify the directory to search via ioVRefNum or ioDirID.

**Hint**: The ioDirID of the root directory of a volume is always 2.

**Hint**: When indexing with **PBGetCatInfo**, ioDirID gets overwritten by each call, so you should set it back to the desired directory ID in each iteration of the loop.

ioFDirIndex should be positive if you are making indexed calls to **PBGetCatInfo**. If ioFDirIndex is 0, you will get information about files or directories, depending on what is specified by ioNamePtr. If ioFDirIndex is -1, you will get information about directories only, and the ioNamePtr field will be ignored.  Notice that if ioNamePtr is ignored, offset 48 contains the dirID of the directory specified by the ioVRefNum that was passed in, and offset 100 contains the parent ID of that directory.

If you specify a full pathname in ioNamePtr, then the call returns information about that path, whether it is a directory or a file. The ioVRefNum field is ignored (but if the full pathname specifies a file, the ioVRefNum field is updated to refer to the file).

If no ioDirID is specified (ioDirID is set to zero), calls to **PBGetCatInfo** will return information about a file in the specified directory, but, if no such file is found, it will continue down the PMSP. The PMSP is not used if ioFDirIndex is non-zero (either -1 or > 0). The default PMSP includes the directory specified by ioVRefNum (or, if ioVRefNum is 0, the default directory) and the directory that contains the System File and the Finder-blessed folder.

The point of this discussion about the PMSP is that you should be careful to make sure that the file you are getting is in the directory you think it is, not in a directory further down the PMSP.

Summary of ioFDirIndex (with DirID = 0 in all cases):

if ioFDirIndex = 0:

Information will be returned about files

Information will be returned about directories as follows:

If a partial pathname is specified in ioNamePtr then the volume and directory will be taken from ioVRefNum.

If a full pathname is specified by ioNamePtr (in which case ioVRefNum will be ignored).

if ioFDirIndex = -1:

Only directory information will be returned.

The name pointed to by ioNamePtr is ignored.

If DirID and ioVRefNum are 0, you will get information about the default directory.

Upon return, examine the ioFlAttrib field to determine whether the entry is for a file or a directory.  If bit 4 is set (0x0010), the entry is a directory.  In that case, the parameter block is returned as a DirInfo structure; otherwise, it's an HFileInfo structure.  The easy way to handle this is to allocate a CInfoPBRec union and create two types of pointers to the data area:

```
CInfoPBRec     pb;                          // 108-byte area
HFileInfo   *fpb = (HFileInfo *)&pb;     // two pointer types
DirInfo         *dpb = (DirInfo *)&pb;
OSErr       rc;

rc=PBGetCatInfo( &pb, FALSE );
if ( fpb->ioFlAttrib & 16) {
    /* . . . it's a directory, use dpb->fieldname . . . */
} else {
    /* . . . it's a file, use fpb->fieldname . . . */
}
```

**PBGetCatInfo** is often used to index through all items in a directory. For instance, the following example uses recursion to search the entire directory tree and list all files that are larger than a specified size. When it encounters a directory, the function FindAll() calls itself to descend down that branch of the tree.

**Example**

```
#include <Files.h>
#include <string.h>
#include <stdio.h>

/* prototypes */
void FindAll(long, char *);
void DoIt(HFileInfo *pb);

Str255     filename, temp;                    /* some global buffers */

main()
{
    FindAll( 2L, "HardDisk");                 /* 2 is dirID of root */
    ExitToShell();
}

/*  Recursive procedure: Examine each entry in directory dirID
    If it is a directory, recurse; otherwise, call DoIt()
    The dirName parm is the C-style accumulated full path; used for display only.
*/
void FindAll( long dirID, char *dirName )
{
    CInfoPBRec     cipbr;                          /* local pb */
    HFileInfo      *fpb = (HFileInfo *)&cipbr;     /* to pointers */
    DirInfo        *dpb = (DirInfo *) &cipbr;
    short          rc, idx;
    Str255         dirFullName;

    strcpy( (char *)dirFullName, dirName );        /* local copy */
    printf("  Searching: %s\n", dirFullName );
    fpb->ioVRefNum = 0;                 /* default volume */
    fpb->ioNamePtr = filename;          /* buffer to receive name */

    for( idx=1; TRUE; idx++) {          /* indexing loop */
        fpb->ioDirID = dirID;           /*  must set on each loop  */
        fpb->ioFDirIndex = idx;

        rc = PBGetCatInfo( &cipbr, FALSE );
        if (rc) break; /* exit when no more entries */

        PtoCstr( filename );            /* make ASCIIZ for printf() */
        if (fpb->ioFlAttrib & 16) {
            sprintf( (char *)temp,"%s:%s", dirFullName,filename );
            FindAll( dpb->ioDrDirID, (char *)temp );/* recursive call */
        }
        else {
            DoIt( (HFileInfo *)&cipbr );
        }
    }
}

/* == Called for every file entry found */
/* == Checks if data+resource size > 64K, and displays info if so */

void DoIt( HFileInfo   *pb)
{
    long    flen;

    flen = pb->ioFlLgLen + pb->ioFlRLgLen;        /* data fork + rsrc fork */
    if ( flen >= 65536 )
        printf( "%8ld %s\n",flen, pb->ioNamePtr );    /* is already ASCIIZ */
}
```