PBGetFInfo Query file date/time, attributes, type, location...

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

```
OSErr       PBGetFInfo(pb, async);
ParmBlkPtr  pb ;        address of an 80-byte FileParam structure
Boolean     async ;     0=await completion; 1=immediate return
            returns     Error Code; 0=no error
```

PBGetFInfo obtains a variety of information about one file or all files in a directory or flat volume. It does not return information about subdirectories.

pb is the address of an 80-byte FileParam structure. The relevant fields are as follows:

Out-In	Name	Type	Size	Offset	Description
->	ioCompletion	ProcPtr	4	12	Completion routine address (if *async* =TRUE)
->	ioVRefNum	short	2	22	Volume, drive, or working directory reference
->	ioFVersNum	SignedByte	1	26	Version (best to use 0)
->	ioFDirIndex	short	2	28	Index (use 0 if not indexing)
<->	ioNamePtr	StringPtr	4	18	Address of full or partial path/filename
<-	ioResult	OSErr	2	16	Error Code (0=no error, 1=not done yet)
<-	ioFRefNum	short	2	24	File access path reference number
<-	ioFlAttrib	SignedByte	1	30	File Attribute bits (locked, directory, etc)
<-	ioFlVersNum	SignedByte	1	31	File version (best to use 0)
<-	ioFlFndrInfo	FInfo	16	32	(File type, creator, flags, icon position, etc.)
<-	ioFlNum	long	4	48	'Hard' file number
<-	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 (seconds since 1/1/1904)
<-	ioFlMdDat	long	4	76	Date/Time of last modification

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
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

Notes: **PBGetFInfo** can be used to obtain information about one particular file or about all files in a directory, as follows:

One File

Set ioFDirIndex to 0. Set ioNamePtr to point to the full or partial filename. If you use a partial pathname, ioVRefNum must identify the root portion of the full pathname (e.g., a working directory reference). On flat volumes, set ioFVersNum to the file's version number (usually 0).

If the file is currently open, the reference number of its first access path is returned in ioFRefNum.

Indexing Through a Directory

Set ioVRefNum to the volume (or working directory) reference number of the directory you wish to peruse. Set ioNamePtr to point to a 32-byte minimum buffer to receive a one-element filename. Iterate ioFDirIndex starting at 1 and continuing until the function returns fnfErr.

If the indexed file is open, the reference number of its first access path is returned in ioFRefNum. In any case, ioNamePtr is filled with the one-element name of the file. You may wish to place this name into a List Manager cell via **LSetCell**.

Note that when indexing, you cannot specify the directory by its name; the ioNamePtr field is used only as a return buffer.

Use **PBGetCatInfo** if you are also interested getting information about subdirectories. **PBHGetFInfo** is similar to **PBGetFInfo**, but it allows you to specify the directory by its 'hard' ID number. The high-level **GetFInfo** returns a small subset of this data - only the Finder information (i.e., the data received in ioFlFndrInfo). All of these information-providers will work on files that are invisible to the Finder.

The meaning of the bits in the fdFlags field of the FInfo structure has changed since System 6.x. Be sure to check the FInfo structure to be sure that the meaning of the bit that you are checking has not changed. For instance, there is no longer a bit in this field that indicates whether a file is locked. To determine whether or not a file is locked, examine the ioFlAttrib field.

You can use **SetFInfo** or **PBSetFInfo** to update Finder information and date/time fields.

Use **Secs2Date** to convert ioFlCrDat and ioFlMdDat into meaningful values. Use **NumToString** to format the file size into readable information.

The following example illustrates file indexing - it lists all files in a given directory. In the example, ioVRefNum is set to 0, specifying the current default directory (see **PBSetVol**), but you could use a volume reference or working directory reference number.

Example

```
#include <Files.h>
#include <OSUtils.h> /* for DateTimeRec structure */

FileParam       fpb;
Str255          filename;
short           j, rc;
```

```
char            *cp = (char *) &fpb.ioFlFndrInfo.fdType;
long            totBytes;
DateTimeRec     dtr;

printf("idx  Name                    Size     Attr  Type  Modified\n");

for( j=1; TRUE; j++ ) {
    fpb.ioNamePtr=filename;
    fpb.ioFDirIndex=j;                      /* the index */
    fpb.ioVRefNum=0;                        /* using current default volume */
    rc = PBGetFInfo( &fpb, FALSE );
    if ( rc ) break;                        /* loop exit */

    PtoCstr( fpb.ioNamePtr );               /* convert to C string for printf */
    Secs2Date( fpb.ioFlMdDat, &dtr );       /* convert date/time */
    printf("%2d  %-31s %8ld %04xH %c%c%c%c %d/%d/%d  %d:%d.%d\n",
        j,
        fpb.ioNamePtr,
        fpb.ioFlLgLen + fpb.ioFlRLgLen,
        fpb.ioFlAttrib,
        cp[0], cp[1], cp[2], cp[3],                 /* 4 character file type */
        dtr.month, dtr.day, dtr.year-1900,
        dtr.hour, dtr.minute, dtr.second
    );
    totBytes += fpb.ioFlPyLen + fpb.ioFlRPyLen;
}
printf( "%5d file(s) occupying %ld bytes disk space\n", j-1, totBytes );
```