PBGetEOF Obtain logical size of an open file

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

```
OSErr       PBGetEOF(pb, async);
ParmBlkPtr  pb;      address of a 50-byte IOParam structure
Boolean     async;   0=await completion; 1=immediate return
            returns  Error Code; 0=no error
```

PBGetEOF returns the end-of-file value (the logical size, in bytes) of an open file.

pb is the address of a 50-byte IOParam structure. The following fields are relevant.

Out-In	Name	Type	Size	Offset	Description
->	ioCompletion	ProcPtr	4	12	Completion routine address (if *async* =TRUE)
->	ioRefNum	short	2	24	File reference number
<-	ioResult	OSErr	2	16	Error Code (0=no error, 1=not done yet)
<-	ioMisc	Ptr	4	28	Size of file, in bytes (must cast long as Ptr)

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
extFSErr	(-58)	External file system
fnOpnErr	(-38)	File not open
ioErr	(-36)	I/O error
rfNumErr	(-51)	Bad ioRefNum

Notes: You might use this before performing a sequential file read in order to know when to stop reading.

The size of the file is returned in ioMisc. Since that structure field is declared as a type Ptr, you will need to cast it as a long integer before attempting arithmetic operations:

```
IOParam  pb;
long     fileSize;
OSErr    rc;

pb.ioRefNum = myFileRef;
rc = PBGetEOF( &pb, FALSE );
if( rc ) { /*. . . handle error . . .*/ }
fileSize = (long)pb.ioMisc;
if (fileSize > 1000) { /*. . . etc. . .*/ }
```

The file will probably occupy more disk space than would seem to be indicated by the file size -- that's because disk space is allocated in blocks. Use **PBGetVInfo** to learn the size, in bytes, of the allocation blocks for a particular volume. Use **PBGetFInfo** to obtain the physical EOF of an open file.