PBLockRange Prevent access to a portion of a shared file

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

```
OSErr       PBLockRange(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
```

On a file opened for shared read/write access, use **PBLockRange** to lock a specified portion. This prevents any concurrent processes from writing to that part of the file. Use **PBUnlockRange** to release the lock.

pb is the address of a 50-byte ioParam structure. The relevant fields are as follows:

Out-In	Name	Type	Size	Offset	Description
->	ioCompletion	ProcPtr	4	12	Completion routine address (if *async* =TRUE)
->	ioRefNum	short	2	24	File reference number
->	ioReqCount	long	4	36	Size of region to lock, in bytes
->	ioPosMode	short	2	44	Positioning Mode (1=absolute, 2=from EOF, etc.)
->	ioPosOffset	long	4	46	Positioning delta (bytes from start, EOF, et.al)
<-	ioResult	OSErr	2	16	Error Code (0=no error, 1=not done yet)

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
eofErr	(-39)	End of file
extFSErr	(-58)	External file system
fnOpnErr	(-38)	File not open
ioErr	(-36)	I/O error
paramErr	(-50)	Range size is less than 0
rfNumErr	(-51)	Bad ioRefNum value

Notes: **PBLockRange** and **PBUnlockRange** are currently only properly implemented for shared volumes (i.e., AppleShare, TOPS). Calling **PBLockRange** or **PBUnlockRange** currently has absolutely no effect for local HFS (or MFS) volumes.

PBLockRange is by networked and multitasking applications that wish to prevent database access collisions. It is an imperfect tool, since a locked region may still be read by another process (thus, a database query at the wrong time will receive information that will be obsolete in a microsecond or two).

The fields of the parameter block match up with those used by **PBRead** and **PBWrite**, making this simple to use. Remember to set the ioPosOffset field back to the start of the locked region after read/write operations, as illustrated below.

PBLockRange is not supported by the 64K ROM File Manager.

Use **PBSetFLock** to lock the whole file or use **PBSetVInfo** to lock an

entire volume.

Example

```
#include <Files.h>

OSErr  rc;
IOParam    pb;
MyStuff    theBuf;             // a fictitious 74-byte structure

pb.ioNamePtr = (StringPtr)"\pMyFile";
pb.ioVRefNum = 0;
pb.ioVersNum = 0;                       // always best to use 0
pb.ioMisc = 0;                      // use volume buffer
pb.ioPermssn = fsRdWrShPerm;      // share read/write access
rc=PBOpen( &pb, FALSE );          // synchronous operation
if ( rc ) { /* . . . handle the error . . . */ }


pb.ioReqCount = 74;                     // assume a 74-byte record
pb.ioPosMode = fsFromStart;         // absolute positioning
pb.ioPosOffset =  74 * 1003;        // access the 1004-th record
pb.ioBuffer = &theBuf;                  // prepare for the read operation
rc=PBLockRange( &pb, FALSE );       // lock; nobody can write
if ( rc ) { /* . . . handle the error . . . */ }
rc=PBRead( &pb, FALSE );            // read the record
if ( rc ) { /* . . . handle the error . . . */ }

    /* (update the record in memory) */

pb.ioPosOffset =  74 * 1003;        // point back to start of record
rc=PBWrite( &pb, FALSE );               // write the changed data
if ( rc ) { /* . . . handle the error . . . */ }
pb.ioPosOffset =  74 * 1003;        // point back to start of  record
rc=PBUnlockRange( &pb, FALSE );         // unlock; now anyone can write
if ( rc ) { /* . . . handle the error . . . */ }
```