FSWrite Write data from memory to a file

#include <Files.h> **File Manager**

```
OSErr   FSWrite(fRefNum, inOutCount, buffer );
short   fRefNum ;      file reference as obtained from FSOpen
long    *inOutCount ;  bytes to write; receives bytes written
Ptr     buffer ;       address of buffer containing data to write
        returns        Error Code; 0=no error
```

FSWrite writes data from memory to an open file or an open device driver. Descriptions of both uses of **FSWrite** are given below

When writing to a file, the data is written starting at the current file mark and advances that mark by the actual number of bytes written.

fRefNum is the reference number of an open file. See **FSOpen** and **OpenRF**.

inOutCount is the address of a positive long integer. On entry, it specifies the number of bytes you wish to write. Upon return, it will contain the actual number of bytes written.

buffer is the address of a memory area containing the data to be written to disk (it should be at least *inOutCount* bytes long).

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

noErr	(0)	No error
dskFulErr	(-34)	Disk full (incomplete or no write)
extFSErr	(-58)	External file system
fLckdErr	(-45)	File is locked
fnOpnErr	(-38)	File not open
ioErr	(-36)	I/O error
paramErr	(-50)	*inOutCount* was negative
rfNumErr	(-51)	Bad file reference number
wPrErr	(-44)	Diskette is write-protected
wrPermErr	(-61)	Write permissions doesn't allow writing

Notes: You can call **FSWrite** repeatedly to write the file sequentially, or you can use **SetFPos** before the write to store data to any part of the file. The file mark is updated (in preparation for the next sequential write).

The data is transferred to the volume buffer and may not be written to disk for some time (even after the file is closed). Use **FlushVol** or **PBFlushFile** to force buffered data to be written to disk.

In the event the disk fills up during the read, the return code will be dskFulErr and *inOutCount* value will contain the actual number of bytes written (probably less than the amount requested on entry).

Use the low-level **PBWrite** function for asynchronous writing and a possible performance increase.

The following example illustrates how to open a file and append a 24-byte data block to the end of the file. See **OpenRF** for an example that uses **FSRead** and **FSWrite** to copy an entire file. See **Using FSWrite to Write to a File** for a longer example.

Example

```
#include <Files.h>
#include <Memory.h>
#define REC_SIZE 24

short   fRef, rc;
long    count;
char    myData[REC_SIZE];                   /* contains data to be written */

rc = FSOpen( "\pHardDisk:MyFile", 0, &fRef );
if ( rc ) { /* . . . handle the error . . . */ }

/* ====== position mark to the end of the file ====== */
rc = SetFPos( fRef, fsFromLEOF, 0 );
if ( rc ) { /* . . . handle the error . . . */ }

count = REC_SIZE;
rc = FSWrite( fRef, &count, myData );
if ( rc ) { /* . . . handle the error . . . */ }

FSClose( fRef );
```

FSWrite can also be used to write data from memory to an open device driver.

refNum is the reference number of an open device driver. See **OpenDriver**.

count is the address of a positive long integer. On entry, it specifies the number of bytes you wish to write. Upon return, it will contain the actual number of bytes written.

buffer is the address of a memory area containing the data to be written to the open device driver (it should be at least *count* bytes long).

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

noErr	(0)	No error
writeErr	(-20)	No response from driver to Write call
badUnitErr	(-21)	*refNum* doesn't match unit table
unitEmptyErr	(-22)	*refNum* specifies NIL handle in unit table
notOpenErr	(-28)	Driver not open

Notes: This procedure, along with **FSRead**, is also used by the **File Manager** to read and write files.