Apple III

SOS

Reference Manual, Volume 2

Apple III

SOS Reference Manual

Volume 2: The SOS Calls

Acknowledgements

Knuth, *Fundamental Algorithms: The Art of Computer Programming*, Vol. I, 2/e.© 1981. Reproduction of book cover. Reprinted with permission.

Writer: Don Reed

Contributions and assistance: Bob Etheredge, Tom Root, Bob Martin, Dick Huston, Steve Smith, Dirk van Nouhuys, Ralph Bean, Jeff Aronoff, Bryan Stearns, Russ Daniels, Lynn Marsh, and Dorothy Pearson

Contents

Volume 2: The SOS Calls

Figures and Tables

Volume 2: The SOS Calls

Preface

Volume 2: The SOS Calls comprises the remaining chapters and the appendixes of this manual. The chapter numbers continue the sequence of those in Volume 1.

Volume 2 defines the individual SOS calls. Chapter 9 contains a description of each file call; Chapter 10, each device call; Chapter 11, each memory call; and Chapter 12, each utility call. Each of these chapters is divided into two sections: calls, and errors.

The calls defined in each chapter are arranged in numerical order by call number (for example, CREATE is $CØ). Each call description contains the following information:

- Definition of the call
- Required parameters
- Optional parameters
- Comments
- Errors

The parameter fields are of four types:

- Pointer (2 bytes): The location of a table or parameter list.
- Value (1, 2, or 4 bytes): A parameter passed by the caller to SOS.
- Result (1, 2, or 4 bytes): A parameter returned by SOS to the caller.
- Value/result (1, 2, or 4 bytes): A parameter passed to SOS and back to the caller, possibly changed.
- Unused (any length): Occurs when the same parameter list is used by two calls, one of which ignores some parameters in the list. An unused field can be of any length.

Each SOS call has three parts, described in Chapter 8 of Volume 1:

- The call block
- The required parameter list
- The optional parameter list

They can be diagrammed as shown in Figure 0-1:

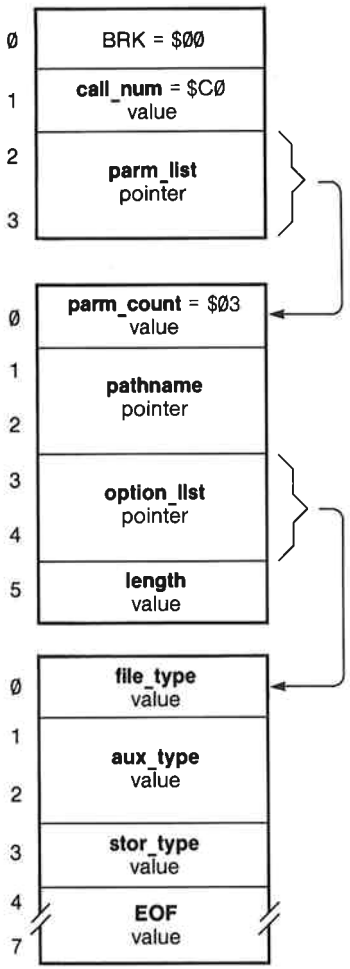


Figure 0-1. Parts of the SOS Call

Each call description is accompanied by a diagram like that shown in Figure 0-1. Most of the diagrams omit the call block, as these are identical, except for the **call_num** , and show only the required and optional parameter lists. In addition, the **parm_count** (shown in the diagram) is omitted from the required parameter list.

The one exception to this pattern is TERMINATE, for which the call block only is shown, as in Figure 0-2, because it differs from the standard form. See section 12.1.6 for details.

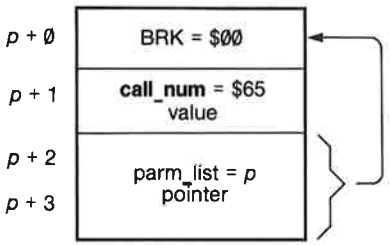


Figure 0-2. TERMINATE Call Block

9

File Calls and Errors

9.1 File Calls

This section contains descriptions of all calls that operate on files. These calls operate on closed files and refer to a file by its pathname.

$C0: CREATE
$C1: DESTROY
$C2: RENAME
$C3: SET_FILE_INFO
$C4: GET_FILE_INFO
$C5: VOLUME
$C6: SET_PREFIX
$C7: GET_PREFIX
$C8: OPEN

These calls operate on access paths to open files and refer to the access path by its **ref_num**, returned by the OPEN call.

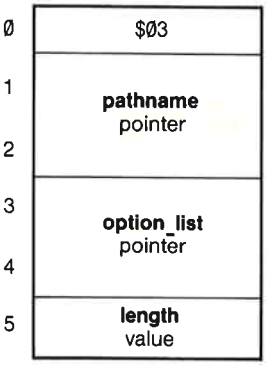
$C9: NEWLINE
$CA: READ
$CB: WRITE
$CC: CLOSE
$CD: FLUSH
$CE: SET_MARK
$CF: GET_MARK
$D0: SET_EOF
$D1: GET_EOF
$D2: SET_LEVEL
$D3: GET_LEVEL

9.1.1 CREATE File Call $CØ

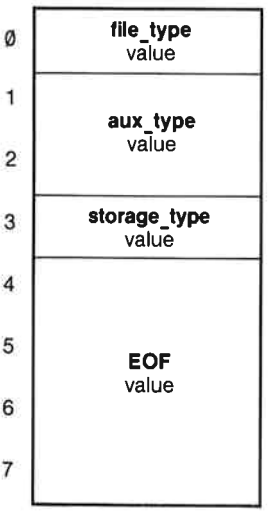
This call creates a standard file or subdirectory file on a volume mounted on a block device. A directory entry is established, and at least one block is allocated on the volume.

This call cannot create a volume directory or a character file. Volume directories are "created" by the formatting utility on the Apple III Utilities disk. Character files are "created" by the System Configuration Program.

CREATE $CØ



Required Parameter List



pathname: pointer

This parameter is a pointer to a string in memory containing the pathname of the file to be created: the first byte of the string contains the number of bytes in the pathname; the remaining bytes contain the pathname itself. The last name in the pathname should be that of a file that does not currently exist in the specified directory, or a DUPERR will result.

option_list: pointer

This is a pointer to the optional parameter list if **length** (below) is between 1 and 8; otherwise it is ignored.

length: 1 byte value
Range: $Ø..$Ø8

This is the length in bytes of the optional parameter list. It specifies which optional parameters are supplied.

The values below tell the number of bytes in a list with complete parameters. If SOS receives an intermediate value, it does not take half a parameter, but reduces the **length** to the next defined value.

Ø = no optional parameters
1 = **file_type**
3 = **file_type** through **aux_type**
4 = **file_type** through **stor_type**
8 = **file_type** through **EOF**

Optional Parameter List

file_type: 1 byte value
Range: $ØØ..$FF
Default: $ØØ

This is the type identifier for this file. The **file_type** does not affect the way in which SOS deals with the file: it is used only by interpreters to determine the internal arrangement and meaning of the bytes in the file. These values of **file_type** are now defined:

$ØØ = Typeless file (BASIC or Pascal "unknown" file)
$Ø1 = File containing all bad blocks on the volume
$Ø2 = Pascal or assembly-language code file
$Ø3 = Pascal text file
$Ø4 = BASIC text file; Pascal ASCII file
$Ø5 = Pascal data file
$Ø6 = General binary file
$Ø7 = Font file
$Ø8 = Screen image file
$Ø9 = Business BASIC program file
$ØA = Business BASIC data file
$ØB = Word Processor file
$ØC = SOS system file (DRIVER, INTERP, KERNEL)
$ØD, $ØE = SOS reserved
$ØF = Directory file (see **storage_type**)
$1Ø..$DF = SOS reserved
$EØ..$FF = ProDOS reserved

aux_type: 2 byte value
Range: $ØØ..$FFFF
Default: $ØØØØ

This is the auxiliary file identifier. It is used by interpreters to store any additional information about the file. BASIC, for example, uses this field to store the record size of its data files. If the file is a volume directory (**storage_type** is $ØF), these bytes contain the total number of blocks on the volume.

storage_type: 1 byte value
Range: $Ø1..$ØD
Default: $Ø1

This indicates whether the file is to be a standard file ($Ø1) or a subdirectory file ($ØD). All other values are illegal and will result in a TYPERR.

EOF: 4 byte value
Range: $ØØØØØØØØ..$ØØFFFFFF
Default: $ØØØØØØØØ

This specifies the amount of space to preallocate for the file.One data block is automatically allocated regardless of the value of **EOF**; additional data blocks are allocated until the number of bytes in the allocated data blocks equals or exceeds **EOF**. In addition to the data blocks, index blocks are allocated as necessary.

The maximum creation size for standard files is $ØØFFFFFF, or $8ØØØ blocks. The maximum creation size for subdirectories is $ØØØØFFFF, or $8Ø blocks. The total number of blocks occupied by a file is the number of data blocks plus the number of index blocks: see Chapter 5 of Volume 1 for more information.

Comments

The file created must be a block file. The **access** attribute of the file is implicitly set to the following:

standard file = $E3: (destroy, backup, rename, write, read)
subdirectory = $E1: (destroy, backup, rename, NO write, read)

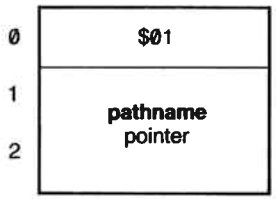
Errors

$27:	IOERR	I/O error
$2B:	NOWRITE	Volume is write-protected
$40:	BADPATH	Invalid pathname syntax
$44:	PNFERR	Path not found
$45:	VNFERR	Volume not found
$46:	FNFERR	Subdirectory file not found
$47:	DUPERR	Attempt to CREATE an existing file
$48:	OVRERR	Overrun error. Either EOF too large or not enough disk space
$49:	DIRFULL	Directory is full
$4B:	TYPERR	**Storage_type** parameter neither $01 nor $0D
$52:	NOTSOS	Not a SOS volume
$53:	BADLSTCNT	Invalid **length** parameter
$58:	NOTBLKDEV	Not a block device

9.1.2 DESTROY File Call $C1

This call deletes the file specified by the **pathname** parameter by removing the file's directory entry. DESTROY releases all blocks used by that file back to free space on that volume.

DESTROY $C1



The file can be either a standard or subdirectory file. Volume directories cannot be destroyed except by physical reformatting of the medium. Character files are "destroyed" by the System Configuration Program.

Required Parameters

pathname: pointer

This parameter is a pointer to a string containing the pathname of the file to be destroyed: the first byte of the string contains the number of bytes in the pathname; the remaining bytes contain the pathname itself.

Comments

A file cannot be destroyed if it is currently open. If the **pathname** refers to a subdirectory file, then that subdirectory must be completely empty in order for the subdirectory to be destroyed.

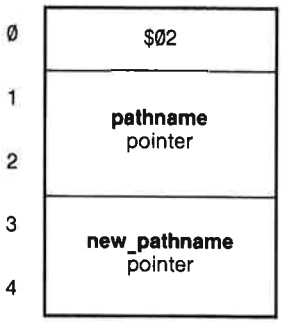
Errors

$27:	IOERR	I/O error
$2B:	NOWRITE	Volume is write-protected
$40:	BADPATH	Invalid pathname syntax
$44:	PNFERR	Path not found
$45:	VNFERR	Volume not found
$46:	FNFERR	File not found
$4A:	CPTERR	Incompatible file format
$4B:	TYPERR	Unsupported file storage type
$4E:	ACCSERR	File's **access** attribute prevents DESTROY
$50:	FILBUSY	File is open. Request denied.
$52:	NOTSOS	Not a SOS volume
$58:	NOTBLKDEV	Not a block device

9.1.3 RENAME File Call $C2

This call changes the name of the file specified by the **pathname** parameter to that specified by **new_pathname**. Only block files may be renamed; character files are "renamed" by the System Configuration Program.

RENAME $C2



Required Parameters

pathname: pointer

This parameter is a pointer to a string containing the old pathname of the file to be renamed: the first byte of the string contains the number of bytes in the pathname; the remaining bytes contain the pathname itself. The **pathname** must refer to either a volume directory, subdirectory, or standard file.

new_pathname: pointer

This parameter is a pointer to a string containing the new pathname of the file to be renamed: the first byte of the string contains the number of bytes in the pathname; the remaining bytes contain the pathname itself. The pathname can be either a complete or partial pathname. Only the last file name of the new pathname may differ from that in the old pathname.

Comments

The file must reside on a block device. Both **pathname** and **new_pathname** must be identical except for the last file name. For example, the path /VOL.1/FILE.1 can be renamed /VOL.1/FILE.2 , but not /VOL.2/FILE.X or /VOL.1/SUBDIR.A/FILE.X .

A file may not be renamed while it is open for writing.

If **new_pathname** matches the pathname of an existing file, you will get a DUPERR.

Errors

$27:	IOERR	I/O error
$2B:	NOWRITE	Volume is write-protected
$4Ø:	BADPATH	Invalid pathname syntax
$44:	PNFERR	Path not found
$45:	VNFERR	Volume not found
$46:	FNFERR	File not found
$47:	DUPERR	Duplicate file name
$4A:	CPTERR	Incompatible file format
$4B:	TYPERR	File storage type not supported
$4E:	ACCSERR	File's **access** attribute prevents RENAME
$5Ø:	FILBUSY	File is open. Request denied.
$52:	NOTSOS	Not a SOS volume
$57:	DUPVOL	Duplicate volume
$58:	NOTBLKDEV	Not a block device