5 Filing System Calls

Any filing system implemented on the Electron offers its facilities by intercepting the standard OS filing system calls. The tape and *ROM filing system code is contained within the operating system ROM. Other filing system software may be implemented in service type paged ROMs. The currently selected filing system must place pointers to relevent routines in the vectors provided for this purpose in page two of memory.

The description of the filing system calls given in this chapter covers a general filing system. The actual implementation will differ slightly between filing systems depending on the suitability of certain calls to their filing system medium.

The filing system calls are:

name	call address	indirection vector
OSFILE	&FFDD	&212
OSARGS	&FFDA	&214
OSBGET	&FFD7	&216
OSBPUT	&FFD4	&218
OSGBPB	&FFD1	&21A
OSFIND	&FFCE	&21C
OSFSC	n/a	&21E

Each of these calls should respond in an appropriate and relevant manner as described in the sections below. Even though the nature of certain filing system's hardware implementation may appear to vary widely, the user is presented with a standard filing system interface wherever possible. Software can be written which functions identically using a number of different filing systems. Where both X and Y are used to point to a parameter block. X holds the low byte and Y holds the high byte of the address.

5.2 OSFILE Read/write entire file or its attributes

Call address &FFDD Indirected through &212

This routine is used to manipulate an entire file. The precise function perfomed by this routine depends on the value in the accumulator. This call can be used to load a file into memory, save a file from memory, delete a file and re-write the file's attributes (e.g. load address, execution address, name etc.). Any information required by the routine to perform its task should be placed in memory. The address of this information should then be passed to the routine in the X and Y registers.

Entry parameters:

A contains a value indicating what action is required
X+Y contain the address of a parameter block

The format of the information placed in the parameter block addressed by X and Y is as follows:

&00 - &01	Address of file name
&02 - &05	Load address of file
&06 - &09	Execution address of file
&0A - &0D	Start address of data (write operations) or Length of file (read operations)
&0E - &11	End address of data (read/write operations) or File attributes (write attributes operation)

The file name should be stored in another part of memory (not sideways ROMs) and be terminated by a carriage return character (&0D) or a space (&20). The least significant byte of the address should be stored in the first of the two bytes. All other parameters are stored in the same order, least significant byte stored first.

The file attributes when required should be provided in the last four bytes of the parameter block. The least significant 8 bits (i.e. the first byte) have the following meanings:

Bit	Meaning if set	
0	not readable	by you
1	not writable	by you
2	not executable	by you
3	not deletable	by you
4	not readable	by others
5	not writable	by others
6	not executable	by others
7	not deletable	by others

The term you here means the originator of the call and the term others means other users of a network filing system.

The action.codes passed to OSFILE in the accumulator have the following effects:

A=0

Save a section of memory as a named file using the information supplied in the parameter block.

A=1

Re-write the catalogue information of an existing file using the information provided in the parameter block. i.e. load and execution addresses.

A=2

Re-write the load address (only) of an existing file identified by the name passed in the parameter block.

A=3

Re-write the execution address (only) of an existing file identified by the name passed in the parameter block.

A=4

Re-write the file attributes (only) of an existing file identified by the name passed in the parameter block.

A=5

Read the named file's catalogue entry and return the file type in the accumulator. These are as follows:

0 returned in A	Nothing found
1 returned in A	File found
2 returned in A	Directory found

A=6

Delete the named file.

A=7

Create a file with a catalogue entry representing the parameter block information but instead of transfering any data pad with null characters.

A=&FF

Load the named file into memory. If the first byte of the execution address field of the parameter block is zero then load to the load address given in the parameter block. If the first byte of the execution address is non-zero then use the file's own load address.

During this call if an error occurs a BRK instruction will be executed which may be trapped if required. During this call interrupts may be enabled but the interrupt status is preserved.

On exit:

A contains the file type
X and Y are preserved
C, N, V and Z are undefined
Information may be written to the parameter block addressed by X+Y.

5.2 OSARGS Read/write open file's attributes
Return current filing system

Call address &FFDA Indirected through &214

This routine is used to manipulate files which are being used for random access. Files used in this way have to be opened using the OSFIND call. When data is being written to or read from the file OSBPUT, OSBGET and OSGBPB can be used but this call should be used to move the sequential pointer used by these calls when data is not transferred. This call is the only way of moving the sequential pointer backwards through a file. OSARGS may also be used to force an update of files onto the medium in use i.e. ensuring that the latest copy of the memory buffer is saved. A number of other functions are performed by this call as detailed below.

Entry parameters:

A contains a value determining the call's actions
X contains a zero page address of a parameter block
Y contains the file handle (see OSFIND) or zero

The parameter block in zero page should be in the user's allocation of zero page. A block of four bytes is required, this will contain the value of the sequential file pointer for read operations or should be set up with a value prior to the call for a write operation. It should be noted that because filing systems should not be languages and so are not copied across to a second processor this parameter block will always exist in the I/O processor even when a Tube is active. If called from the second processor, the parameter block will be copied across into the I/O processor before the filing system is called.

Interrupts may be enabled during a call but the interrupt status will be preserved.

If Y=0 and A=0 then return the current filing system in A.

value returned	filing system
0	no current filing system
1	1200 baud cassette

2	300 baud cassette
3	ROM filing system
4	Disc filing system
5	Econet filing system
6	Telesoftware filing system
7	IEEE filing system
8	ADFS
9	Reserved
10	Acacia RAM filing system

If Y=0 and A=1 then return the address in the I/O processor of the rest of the command line will be returned in the two least significant bytes of the zero page parameter block. This enables software to access the parameters passed with '*' commands.

If Y=0 and A=&FF then update all files onto the filing system medium; this ensures that the medium has the latest copy of the buffers.

If Y is non-zero then the value in Y is assumed to be a file handle (see OSFIND). The value passed in A determines the action on the open file specified by Y

A=0

Read sequential file pointer (written to the zero page parameter block). This pointer is the same as that used by BASIC called PTR#.

A=1

Write sequential file pointer.

A=2

Read length of sequential file. This value is the same as that returned by EXT# in BASIC.

A=3

Write length of sequential file. This call is not implemented in all filing systems but where implemented may be used either to truncate a file or to extend it (in which case it will be padded with zeroes).

A= &FF

Update this file onto the filing system medium.

On exit:

A is preserved except on a call with A=0 and Y=0
X and Y are preserved

C, N, V and Z are undefined

5.3 OSBGET Get a single byte from an open file

Call address &FFD7 Indirected through &216

This routine returns the value of a byte read from a file opened for random access. The file should have been previously opened using OSFIND, The file handle required by this call will have been provided from this OSFIND call.

Entry parameters:

Y contains file handle

A byte is read from that point in the file determined by the sequential file pointer. During each call of OSB GET the sequential file pointer is incremented by one. Thus successive OSBGET calls can be used to read bytes from the file sequentially. This pointer may be read or written using the OSARGS call thus enabling the use of random access.

Interrupts may be enabled during a call but the interrupt status will be preserved.

A is returned containing the value of the byte read

On exit:

X and Y are preserved
C=1 if the end of file was reached i.e. invalid call, in which case A=&FE.

N, V and Z are undefined

5.4 OSBPUT Write a single byte to an open file

Call address &FFD4 Indirected through &218

This call is the complement to the OSBGET call described above. A file handle should be provided from a prior OSFIND call and the sequential file pointer is used to locate the point in the file where the byte is written.

Entry parameters:

A contains the byte to be written to the file.
Y contains the file handle.

During the call a byte will be written to the file and the sequential pointer will be incremented. If the sequential file pointer reaches the end of the file the file will be extended to accommodate any new data written where possible.

Interrupts may be enabled during a call but the interrupt status will be preserved over a call.

On exit:

A, X and Y are preserved

C, N, V and Z are undefined

5.5 OSGBPB Read/write a group of bytes to/from an open file

Call address &FFD1 Indirected through &21A

This routine enables the transfer of a group of bytes to or from an open file. This routine is implemented particularly for filing systems which have a high time overhead associated with each data transfer e.g. the Econet.

Entry parameters:

A contains a value which determines the action performed
X+Y contain a pointer to a parameter block in memory

The parameter block should contain information in the following format:

&00	file handle
&01 - &04	address of data for transfer
&05 - &08	number of bytes to transfer
&09 - &0C	sequential file pointer to be used

The bytes in each parameter should be placed least significant byte first.

The address should include a high order address (see OSBYTE &82) to indicate if the data is in an i/o processor or a second processor.

The sequential file pointer passed in the parameter block will only replace the old value of this pointer when appropriate.

The action codes passed to the routine will have the following effects:

A=1

Write a group of bytes to the open file. The sequential pointer given will indicate the point in the file where these bytes are put and this pointer will be incremented by the number of bytes written.

A=2

Write a group of bytes to the open file without using the sequential file pointer value given in the parameter block. The existing value of the pointer will mark the point in the file where these bytes are put and the pointer will then be incremented by the number of bytes written.

A=3

Read a group of bytes from an open file. The sequential pointer given in the parameter block will indicate where the bytes should be read from within the file. The pointer will then be incremented by the number of bytes read.

A=4

Read a group of bytes from an open file disregarding the sequential file pointer value given in the parameter block. The existing pointer value will be used and subsequently incremented by the number of bytes read.

A=5

Return the title associated with the currently active medium and return boot/startup attribute, This information is written to the address pointed at by the X andY registers. The format of the data is:

&00	n, the length of the title string
&01 - n + 1	the title string, ASCII characters
n+2	value indicating boot/start up options

The start up information is filing system dependent.

A=6

Return the currently selected directory and device identity. Two items of data are written to the parameter block. The format of the data is:

&00	n, the length of the directory name
&01 - n+1	directory name, ASCII string
n + 2	m, the length of the device identity
n + 3 - n + m + 3	the device identity

A=7

Read the currently selected library, and device, The data format is the same as that used for A=6.

A=8

This call is used to read file names from the current directory. The parameter block should be set up so that the number of file names to transfer is placed in the 'No. of bytes to transfer' field, For the first call the 'sequential file pointer' field should be set to zero. The sequential file pointer is incremented each time this call is made so that it points to the next file name for transfer.

The data is transferred to the specified address in the form of a list of file names. Each file name takes the form of an ASCII string preceded by a single byte value indicating the length of the string. The number of filenames in this list is determined by the value passed in the parameter block unless the end of the directory is reached.

This call also returns a cycle number in the 'file handle' field of the parameter block. This cycle number represents the number of times the current catalogue has been written to.

Exit conditions:

A, X and Y are preserved
N, V and Z are undefined
C=1 if the transfer could not be completed

In the event of a transfer not being completed the parameter block contains the following information:

(a) the number of bytes or names not transferred in the 'number of bytes to transfer' field

(b) the 'address' field contains the next location of memory due for transfer

(c) the 'sequential pointer' field contains the sequential file pointer value indicating the next byte in the file due for transfer

5.6 OSFIND Open or close file for random access

Call address &FFCE Indirected through &21C

This call is used to allocate a file handle for subsequent use by OSARGS, OSBGET, OSBPUT and OSGBPB. This call should also be used to close a file when no further access is required. In this instance the file handle is released for re-allocation and the file medium is updated from the buffers in memory.

The file handle is a single byte value which uniquely identifies an open file. This provides a less cumbersome method of addressing the file in question than using the filename each time. The number of files which can be open at any one time is filing system dependent. The actual range of handle values allocated by each filing system is different. The ranges which have been allocated by Acorn are listed under OSFSC with A=&07.

Entry parameters

(a) To open a file

The accumulator contains a code indicating the type of access for which the file should be opened:

A=&40 input only
A=&80 output only (i.e. will attempt to delete file first)
A=&C0 input and output

X and Y contain the address of a file name string (low byte, high byte). The filename string should be terminated by a carriage return character (&0D).

The accumulator will be returned containing the file handle which has been allocated or zero if the file could not be opened. Note that if the filename is syntactically bad, or involves a non-existent directory, a BRK 'Not found' error may occur.

(b) To close a file

A=0 Y contains the handle of the file to be closed or Y=0 to close all currently open files.

On exit:

A returns file handle on opening otherwise preserved
X and Y are preserved

C, N, V and Z are preserved
Interrupts may be enabled during call, status preserved

5.7 OSFSC Miscellaneous filing system control

No OS call address Indirected through &21E

This vector contains an entry point into the current filing system which may be used to invoke a number of miscellaneous filing system functions. Because there is no direct call address this call can only be made from within an I/O processor and is not

available for code running on a second processor. However many of the facilities are indirectly available via other OS calls which subsequently make calls through this vector.

Entry parameters:

The accumulator contains an action code determining which control function is performed.

A=0 *OPT command

The operating system makes this call in response to '*OPT' being submitted to the command line interpreter or in response to OSBYTE &8B. X andY contain the parameters passed with the '*OPT' command.

A=1 Check for end of file (EOF)

This call is made by the operating system in response to OSBYTE &7F. The call is entered with a file handle value in the X register. The X register should be returned containing the value &FF if an EOF condition exists, otherwise it should be returned containing zero.

A=2 '*/' command

The filing system should attempt to *RUN the file whose name follows the '/' character. The operating system command line interpreter will make this call in response to a command begining '*/'. The X andY registers contain the address of the file name string (not including the '*/' characters).

A=3 Unrecognised *command

The operating system issues this call when an unrecognised command has been submitted to the command line interpreter. This call is made after the 'unrecognised *command' paged ROM service call has been made (see paged ROMs section 10.1). The command name string is addressed by the X and Y registers.

Filing systems will respond to this call by attempting to *RUN the file having the command name. The idea behind this is to enablethe implementation of command like utilities which are stored on the filing system media. However in the case of a filing system being unable to execute the file without delay the filing system should respond to this call with a 'Bad Command' message instead.

A=4 *RUN attempted

The operating system passes on the file name given with a *RUN command to the current filing system using this call. The X and Y registers contain the address of the file name string, The filing system should then load and execute the code in this file.

A=5 *CAT attempted

This call is made by the operating system in response to a *CAT command. The X and Y registers contain the address of the rest of the command string where any parameters required by the routine may be found.

A=6 New filing system selected

This call is issued when the current filing system is being changed. The deselected filing system should respond by closing any *SPOOL or *EXEC files using OSBYTE &77 and prepare itself for the handover.

A=7 Return handle range

This call may be made to determine the range of values allocated as file handles by the currently selected filing system. Below is a list of the handle ranges that have been allocated by Acorn.

filing sysem	handle range, inclusive	
Tape filing system	1 (&01)	2 (&02)
*ROM filing system	3 (&03)	3 (&03)
Teletext filing system	14 (&0E)	15 (&0F)
Disc filing system	17 (&12)	21 (&15)
Network filing system	32 (&20)	39 (&29)

Winchester DFS	48 (&30)	57 (&39)
reserved values	64 (&40)	71 (&49)
Acacia RAM filing system	96 (&60)	101 (&65)
IEEE filing system	240 (&F0)	255 (&FF)

The X register is returned with the lowest value which may be allocated as a file handle and the Y register returned with the highest value used.

A=8 OS *command about to be processed

The operating system makes this call prior to executing a *command. Acorn DFS uses this call to implement the '*ENABLE' protection mechanism. This call may also be used by filing systems to output extra messages e.g. 'Compaction recommended' when free space has become highly fragmented on a disc.

On exit:

Registers returned as described above
Otherwise registers undefined
Status flags undefined
Interrupts may be enabled, status preserved