Chapter 10 : Machine Code Reference Section
BBC MOS Interface

This section is intended as a full and detailed handbook of the machine code interface to the network filing system. It covers the filing system interface, which is (in theory at least) the same as the interface to disc, tape or any other filing system. It also covers the network communication functions which are network specific.

The section assumes knowledge of the BBC operating system, BASIC (including the indirection operators - see chapter 39 of the BBC User Guide), assembler and control blocks.

Because this section is a reference section, it is not necessary to understand all of it at once. Instead it should be possible to look up a certain call as it is needed.

Network Versions

There are three main Network filing system ROMs available for the BBC computer. These are :-
NFS version 3.34
NFS version 3.60
Advanced NFS (Only available for the Master, Econet Terminal & Compact)
(see OSARGS A=2 Y=0 and *HELP for testing the version)

Although they are intended to control the same operations there are some important differences. Because all versions are common, programs should be written so that they will work on all.

Memory Addresses

The filing system assumes that addresses in a remote or local machine are 4 bytes long. For a normal BBC computer this means that the top 16 bits are not relevant, but when a 2nd processor is used only the address &FFFFxxxx specifies the I/O processor. For the 6502 2nd processor the address &0000xxxx specifies the language processor. In general if the I/O processor is the intended destination then &FFFFxxxx should always be used. Unless otherwise specified, multi-byte numbers are stored low byte first.

For an Acorn Master & Acorn Econet Terminal the I/O processor is located at address &FFFFxxxx and the screen RAM is located at address &FFFExxxx.

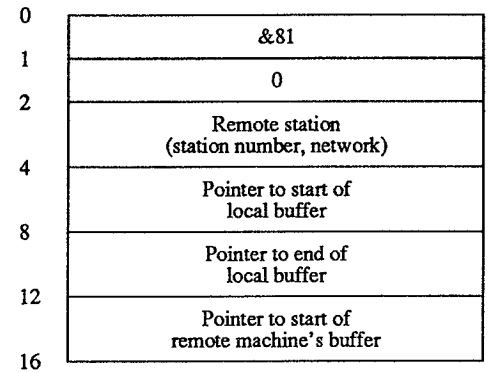
Calling Operating System Routines

These routines handle all the input and output to the Econet system. All routines require the Decimal flag (D) to be clear on entry. The Interrupt flag (I) is always preserved (but interrupts may be enabled during the call).

Some operating system routines require a control block; a small area of memory set up to hold the parameters to be used by the routine. An area of memory must be preserved for the control block, filled with the parameters you want, and pointers to the area given to the routine.

The information for most calls is described here in the form of a control block. The number on the left hand side is refering to the offset from the start of the control block. To set up a control block an area of memory should be reserved before the call is performed. It is simple to use the BASIC indirection operators to set up the values in the control block.

The program listed below performs a peek (i.e. a transfer of data from the memory of a remote machine to the memory of the local machine) by using the following control block :-




```
 10 DIM blk% 15:       REM  Reserve a block of data
 20 INPUT "Station : "station%
 30 blk%?0=&81:        REM  Control byte for peek
 40 blk%?1=0:          REM  Port number for immediate operation
 50 blk%!2=station%:   REM  Insert station number
 60 blk%!4=&FFFF7C00: REM  Start of MODE 7 screen (not scrolled)
 70 blk%!8=&FFFF8000: REM  End of MODE 7
 80 blk%!12=&FFFF7C00:REM  Start of MODE 7 in remote machine
 90
100 X%=blk%:           REM  X points to low byte
110 Y%=X% DIV 256:     REM  Y points to high byte
120 A%=&10:            REM  Set accumulator
130 CALL &FFF1:        REM  Call OSWORD
```

Note that the above program does not check that the transmit operation worked. For a complete description of the peek operation see sections 10.9 and 10.14.

Summary of all calls for Econet

Name	Call Address	Section	Description
OSGBPB	&FFD1	10.1	Read/write group of bytes to a file, read user's environment
OSFIND	&FFCE	10.2	Open/close file for random access
OSBGET	&FFD7	10.3	Get byte from file (Use OSGBPB if possible)
OSBPUT	&FFD4	10.4	Write byte to file (Use OSGBPB if possible)
OSARGS	&FFDA	10.5	Read/write file's arguments (using a handle)
OSFILE	&FFDD	10.6	Read/write file's arguments
OSCLI	&FFF7	10.7	Send string to command line interpreter
OSWORD	&FFF1	10.8	Network Specific commands
		10.9	Transmit data (A=&10)
		10.10	Receive data (A=&11)
		10.11	Read arguments (A=&12)
		10.12	Read/write station information (A=&13)
		10.13	Send to File Server (A=&14)
OSBYTE	&FFF4	10.14	Poll transmission and reception
		10.15	Transmitting
		10.16	Receiving
		10.17	Port numbers
		10.18	The Bridge
		10.19	Printers
		10.20	The File Server Interface
		10.21	Password entry format
		10.22	Application Notes
		10.23	Netmon
		10.24	Econet Protocols

10.1 OSGBPB Call Summary

Entry point &FFD1
Indirected via &21A

On entry,

A=Reason code
YX points to a control block 13 bytes long.

Do not put the control block in page zero. There is a bug in NFS 3.34 such that OSGBPB will not work at all if this is done.

Value in A	Function
A=1	Write block using offset
A=2	Write block ignoring offset
A=3	Read block using offset
A=4	Read block ignoring offset
A=5	Read currently selected disc title and boot option
A=6	Read directory
A=7	Read library
A=8	Read specified number of file names

On exit,

A = Return code indicating whether the requested function is supported by the currently selected filing system. If A=0 then the operation was attempted. A is returned unchanged if that function is not available.
YX undefined

Control block is modified on completion. On some filing systems the carry flag is set on reaching the end of the file/directory, this should not be relied on. To write software that is compatible with all filing systems the only method of finding whether all the files have been read is to look at the contents of (control block + 4).

Write data giving offset

OSGBPB A=1

General description

This call writes a block of data from RAM to a file, specifying where in the file to put the bytes.

On entry,

A=1
YX point to the control block shown below :-

0	File Handle
1	Address of data
5	Number of bytes to transfer
9	Offset in file
13	

On exit,

The sequential pointer (PTR#) will have been updated to point immediately beyond the last byte written.
If the operation succeeded the control block is :-

0	File Handle
1	Old address + Number of bytes transferred
5	Number of bytes not transferred (should=0)
9	Old offset + Number of bytes transferred
13	

Example :

The procedure below uses this call to write a string to the file opened as `handle%`.

```
DEF PROCwrite_string(handle%,$data%,offset%)
 LOCAL A%,X%,Y%
 blk%?0=handle%
 blk%!1=data%
 blk%!5=LEN($data%)
 blk%!9=offset%
 X%=blk%
 Y%=X% DIV 256:A%=1
 IF (USR(osgbpb) AND &FF)<>0 THENPROCsimulate_osgbpb(blk%)
ENDPROC
```

The procedure below can be used on systems that do not support OSGBPB (e.g.TAPE).

```
DEF PROCsimulate_osgbpb(blk%)
 LOCAL I%
 PTR#?blk%=blk%!9
 FOR I%=blk%!1 TO blk%!1+blk%!5
 BPUT#?blk%,?I%
 NEXT
ENDPROC
```

The following program uses the above procedure to create a file called 'filename' and write two strings to it.

```
10 DIM data% 255 :REM Maximum size of my string (actually 256 bytes)
20 DIM blk% 12   :REM My control block
30 osgbpb=&FFD1
50 out_ch%=OPENOUT"Filename"
60 PROCwrite_string(out_ch%,"This is a test",0)
70 PROCwrite_string(out_ch%,"Another piece of text",&22A)
80 CLOSE#out_ch%
90 END
```

Compatibility between Filing Systems :

Not supported on TAPE, ROM or some non-Acorn DFS. Note on the Acorn File Server a write beyond the end of the file will give an EOF (end of file) error.

Write Data

OSGBPB A=2

General Description

This call writes a block of data to a file using the current pointer (i.e. the value returned by PTR#).

On entry,

A=2
YX point to the control block shown below :-

0	File Handle
1	Address of data
5	Number of bytes to transfer
9	Ignored
13	

On exit,

The sequential pointer (PTR#) will have been updated to point immediately beyond the last byte written.
If the operation succeeded the control block is :-

0	File Handle
1	Old location + Number of bytes transferred
5	Amount of bytes not transferred (should=0)
9	Corrupt
13	

Compatibility between Filing Systems :

Not supported on TAPE or ROM or some non-Acorn DFS.

Read Data giving offset

OSGBPB A=3

General description

This call reads a block of data from a file into RAM, from a specified offset in the file (i.e. ignoring PTR#). On reaching the end of the file the carry flag (C) is set, and the amount of data not transferred is returned in the control block. An EOF (End of file) error will be returned if the offset is past the end of the file. If the offset of a file is equal to the extent, then no bytes will be transferred.

On entry,

A=3
YX point to the control block shown below :-

0	File Handle
1	Location to put the data
5	Number of bytes to transfer
9	Offset in file
13	

On exit,

If the operation was successful the control block is :-

0	File Handle
1	Old location + Number of bytes transferred
5	Number of bytes not transferred
9	Old offset + Number of bytes transferred
13	

The number of bytes not transferred will be zero unless the end of file has been reached; in which case all bytes from the offset up to the end of the file will have been transferred, and the number not transferred is the difference between the number originally asked for and the number actually transferred. Note that an area of memory large enough to hold the entire quantity of data asked for will always be corrupted, so it is inefficient to request many more bytes than are expected to be available.

NFS 3.6 does NOT return the correct result for EOF# after this call. The only way of checking to see if the End Of File has been reached, is to read the result from the control block (bytes 5-8 inc.).

Compatability between Filing Systems :

Not supported on TAPE or ROM or some non-Acorn DFS. Note that the Acorn DFS does not corrupt more memory than is required to hold the data loaded, whereas the NFS will always transfer the amount of data requested. Therefore programs which request more bytes than there is memory space to hold may work on the DFS but crash with NFS.

Read Data

OSGBPB A=4

General description

This call reads a block of data to a file, at the position given by the sequential pointer (PTR#).

On entry,

A=4
YX point to the control block shown below :-

0	File Handle
1	Address of data
5	Number of bytes to transfer
9	No data needed but will be modified
13	

On exit,

If the operation was successful the control block is :-

0	File Handle
1	Old location + Number of bytes transferred
5	Number of bytes not transferred
9	Corrupt
13	

N.B.
Number of bytes not transferred and memory corruption as for OSGBPB A=3.

Compatibility between Filing Systems :

Not supported on TAPE on ROM or some non-Acorn DFS. Note that not all filing systems return C=1 to indicate that the end of file has been reached. It is strongly recommended that to find whether the end of file has been read the number of bytes not transferred is checked for a non-zero value.

Read CS disc name

OSGBPB A=5

General description

This call returns your currently selected disc name and your boot option.

On entry,

A=5
YX point to the control block shown below :-

0	xxx
1	Address where data will be loaded
5	xxx
13	

xxx indicates that no parameter is required.

On exit,

C undefined

The control block is left unchanged.
The address pointed to by the control block is modified as shown below :-

0	Length of disc title (n)
1	Disc title (Max.16 characters)
(n+1)	Your boot option as set by *OPT4
(n+2)	

The disc name is normally 16 characters padded with spaces. It is possible for the disc name to have spaces as significant characters, e.g. 'Wombat 2' (although on the network it would not be possible to use this name). Therefore to read the disc title the name should be read backwards, stripping off spaces until the first non-space character.

Compatibility between Filing Systems :

Not supported on TAPE or ROM.

Read CSD name

OSGBPB A=6

General description

This call returns your currently selected directory name.

On entry,

A=6, YX point to the control block shown below :-

0	Undefined
1	Address to put the data
5	Undefined
9	Undefined
13	

There is a bug in NFS version 3.34 which sometimes causes the error 'No reply'. To get around this problem the following fix should be used before calling OSGBPB.

```
.fix_bug  LDA #2:LDY #0:JSR &FFDA \Call OSARGS to read version of NFS
          CMP #2:BNE not_required \Fix not required if not version 3.34
          LDX #block MOD 256      \Fix NFS workspace by writing
          LDY #block DIV 256      \ an &CB to location &B8
          LDA #6:JSR &FFF1        \OSWORD to write to I/O processor
.not_required

                    (Rest of the code)

.block    EQUD &FFFF00B8          \Write &CB to location &FFFF00B8
          EQUB &CB
```

N.B. It is essential that the version is checked before modifying this location, because it cannot be guaranteed that the use of this location will not change.

On exit,

The address pointed to by the control block is modified as shown below :-

0	Length of drive number(v)=0 on Econet
1	ASCII coded drive number This field is not present on the Econet
v	Length of directory name (n) (Usually 10 characters)
(v+1)	Directory name
(n+2)	Ownership (0=Owner,&FF=public)
(n+3)	

The directory title may be padded with spaces.

Example :

The following program will print out the currently selected directory, and drive number, on all filing systems that support the call.

```
 10 REM   Read currently selected directory name
 20 REM  This program is compatible with all filing systems
 30 REM   (C) A.J.Engeham, SJ Research
 40 REM_____________________________________________
 50
 60 osgbpb=&FFD1
 70 DIM blk% 17,buffer% 100
 80 blk%?0=0
 90 blk%!1=buffer%
100 blk%!5=0
110 blk%!9=0
120 X%=blk%:Y%=X% DIV 256
130 A%=6
140 IF (USR(osgbpb) AND &FF)=6 THENP."Not supported":END
150 pos%=0
160 PROCread("Current drive number is : ")
170 PROCread("Currently selected directory is : ")
180END
190
200DEF PROCread(string$)
210 LOCAL V%,result$,I%
220 V%=buffer%?pos%
230 pos%=pos%+1
240 IF V%=0 THENENDPROC
250 FOR I%=pos% TO pos%+V%-1
260 result$=result$+CHR$(I%?buffer%)
270 NEXT
280 pos%=pos%+V%
290 PRINTstring$;result$
300ENDPROC
```

Compatibility between Filing Systems :

Not supported on TAPE or ROM. Note that early versions of ADFS do not return the ownership byte.

Read LIB name

General description

This call returns your currently selected Library.

On entry,

A=7
YX point to the control block shown below :-

0	Undefined
1	Address to put the data
5	Undefined
9	Undefined
13	

On exit,

The address pointed to by the control block is modified as shown below :-

0	Length of drive number(v)=0 on Econet
1	ASCII coded drive number This field is not present on the Econet
v	Length of Library name (n)
(v+1)	Library name
(n+1)	Ownership (0=Owner,&FF=public)
(n+2)	

Example :

See OSGBPB with A=6

The Library name may be padded with spaces.

Compatibility between Filing Systems :

Not supported on TAPE or ROM.

Read objects

General description

This call reads file/directory names from your currently selected directory.

On entry,

A=8
YX point to the control block shown below :-

0	0
1	Location to put the data
5	Maximum number of file names to transfer
9	Which file to start on
13	

On exit,

The control block is modified as shown below.

0	Cycle number
1	Pointer to the end of data
5	Number of files NOT transferred
9	Which file to ask for next
13	

The file names are padded with spaces.

0	Length of filename (n)
1	Filename (padded with spaces)
(n+1)	Length of filename(if asked for)
(n+2)	Filename
	. . .
v	

Example

The program shown below will print out all the file names of your currently selected directory. Note that, because the starting file is the first one in the directory, this program will produce the same results for the Disc Filing System as the Network Filing System.

```
DIM blk% 13,buffer% 20
 blk%!9=0:REM   Start at the first file
REPEAT
 blk%?0=0:REM   Cycle number returned here
 blk%!1=buffer%
 blk%!5=1:REM   Number of files to read
 X%=blk%:Y%=blk% DIV 256
 A%=8:CALL &FFD1
 IF blk%!5=0 THEN?(buffer%+1+buffer%?0)=13:PRINT $(buffer%+1)
UNTIL blk%!5=1
```

Compatibility between Filing Systems :

Not supported on TAPE or ROM. Entry number are not guaranteed to be sequential; the only value which should be used in general are 0 and values returned by a previous call. In the special case of Econet, entry numbers start from zero and ascend in steps on 1 to specify objects in alphabetical order.

The carry flag is NOT valid on exit. The DFS is misleading in this respect, although C happens to be significant on DFS systems, it is not an official Acorn feature.

10.2 OSFIND

Entry point &FFCE
Indirected via &21C

On entry,

A=Type of operation.

On exit,

A=file handle if successful.
A=0 not possible to open file.
C,N,V,Z are undefined.
Contents of control block are preserved.

Value in A	Function
A=0	Close file specified by Y register
A=&40	Open file for input
A=&80	Open file for output (i.e. create new file deleting old file)
A=&C0	Open file for update (i.e. input and output).

Call Summary

Close files

General description

If Y=0 then the call closes all open files but will not close directory context handles.
If Y<>0 then the call closes the file handle, or context handle, held in Y.

This call is the equivalent to CLOSE#Y in BASIC.

On entry,

A=0
Y=handle to close (0 to close all files)

On exit,

A,X,Y undefined

Compatibility between Filing Systems :

Supported on all filing systems. Y=0 does not work on some version of Master DFS.

Open file for input

OSFIND A=&40

General description

Returns a handle for a file whose name is in memory, pointed to by YX. This handle can then be used for reading bytes from that file. The handle can also be used as a context handle to describe your environment on your File Server. This call is equivalent to the BASIC 2 keyword OPENIN.

On entry,

YX point to a control block containing a string terminated by a carriage return (&0D).
A=&40

On exit,

A=file handle if successful
A=0 if unable to open file
X,Y undefined
C,N,V,Z undefined

Compatibility between Filing Systems :

Supported on all filing systems.

Open file for output

OSFIND A=&80

General description

Create a file with the name given at memory address YX and returns a handle for that file. The file is opened for both reading and writing. Any previous file of that name is deleted. This is equivalent to the BASIC keyword OPENOUT.

On entry,

YX point to a filename terminated by a carriage return (&0D).
A=&80

On exit,

A=file handle if successful
A=0 if unable to open file
X,Y undefined
C,N,V,Z undefined

Compatibility between Filing Systems :

Not supported on ROM, or any other read only filing systems.

Open file for update

OSFIND A=&C0

General description

Returns a handle for a file whose name is pointed to by YX. The file is opened for both reading and writing. This is equivalent to the BASIC 2 keyword OPENUP or the BASIC 1 keyword OPENIN.

On entry,

YX point to a control block terminated by a carriage return (&0D).
A=&C0

On exit,

A=file handle if successful
A=0 if unable to open file
X,Y undefined
C,N,V,Z undefined

Example

The example shows how to OPENUP a file for BASIC 1 and BASIC 2.

```
DIM dummy% 80
$dummy%=""
INPUT "Filename : "dummy$
P.FNopenup(dummy$)
END

DEF FNopenup($dummy%)
 LOCAL A%,X%,Y%
 X%=dummy%
 Y%=dummy% DIV 256
 A%=&C0
=(USR(&FFCE) AND &FF)
```

Compatibility between Filing Systems :

Not supported on ROM. TAPE treats the open for update as an open for output (i.e. OSFIND with A=&80).

10.3 OSBGET

Read Byte

Entry point &FFD7
Indirected via &216

General description

This call reads one byte from a file, using a handle returned from an OSFIND call. The file pointer (PTR#) is incremented.

On entry,

Y=file handle

On exit,

X,Y preserved
If successful A=byte retrieved from file, C=0
If the end of the file is reached A=254, C=1
If an attempt to read past the end of file is made, an 'EOF' error will be returned.

Compatibility between Filing Systems :

Supported on all filing systems. Although this call is supported on the Econet it is very slow, it is essential that wherever possible OSGBPB is used. This is because the file server takes time to process each byte, and time is taken to send each byte across the network; if the file server (if it isn't one of our multitasking file servers) is processing someone else's command your machine will wait in the mean time. If *PUTGET is active, or your machine has ANFS, then the indirection vector will be modified so that the data is read from the fileserver using OSGBPB, and later retrieved from a buffer in memory.

10.4 OSBPUT Write Byte

Entry point &FFD4
Indirected via &218

General description

This call puts one byte to a file, using a handle returned from an OSFIND call. The file pointer (PTR#) is incremented.

On entry,

Y=file handle
A=byte to put

On exit,

A,X,Y are preserved
N,V,Z,C are undefined

Compatibility between Filing Systems :

Supported on all filing systems except ROM, and other read only filing systems. Although this call is supported on the Econet it is very slow, it is essential that wherever possible OSGBPB is used. This is because the file server takes time to process each byte, and time is taken to send each byte across the network; if the file server (if it isn't one of our multitasking file servers) is processing someone else's command your machine will wait in the mean time. If *PUTGET is active, or your machine has ANFS, then the indirection vector will be modified so that the data is read from the fileserver using OSGBPB, and later retrieved from a buffer in memory.

10.5 OSARGS Call Summary

Entry point &FFDA
Indirects via &214

On entry,

Value in Y	Value in A	Function
Y=0	A=0	Return filing system type
Y=0	A=1	Return rest of command line
Y=0	A=2	Return version of NFS ROM
Y<>0	A=0	Read sequential pointer
Y<>0	A=1	Write sequential pointer
Y<>0	A=2	Read extent of file
Y<>0	A=3	Write extent of file (ANFS only)
Y<>0	A=4	Read current amount of space allocated to file (ANFS only)
Y<>0	A=&80	Read internal information (ANFS only)
	A=&FF	Ensure file is up to date on the media

On exit,

A=0 if the operation supported (except A=0, Y=0) otherwise A is preserved
X,Y are preserved
C,N,V,Z are undefined

Return Filing System Type

OSARGS A=0 Y=0

General description

This calls returns, in A, the type of your current filing system.

On entry,

A=0
Y=0

On exit,

A=filing system
X,Y undefined
N,V,Z,C undefined

Value in A	Meaning
0	No current filing system
1	Cassette, 1200 baud
2	Cassette, 300 baud
3	ROM
4	Disc
5	Econet
6	Teletext/Prestel
7	IEEE
8	Advanced Disc
9	Host
10	VFS Video disc
16	Acacia RAM

To select filing system 'n' use OSBYTE with A=&8F, X=&12, Y=n.

Compatibility between Filing Systems :

Supported on all filing systems.

End of line parameter

OSARGS A=1 Y=0

General description

This call set the zero page locations (pointed to by X on entry to the call) to point to the parameter at the end of the last command line.

Unfortunately, there is a bug in version 3.34 of the NFS ROM such that the call returns the pointer pointing to the text immediately after the last *. For example, in the command line '********VIEW 23' it would point to the 'V' in VIEW. On all other filing systems that support this call, and later versions of the NFS ROM, the call returns the pointer pointing to the '2' in the 23.

On entry,

A=1
Y=0
X=Zero page location

On exit,

A=0 if call supported otherwise A is preserved.
X,Y preserved
N,V,Z,C undefined

Example :

The program below shows a typical method of getting round the problem with different versions of the NFS.

```
osargs=&FFDA

.error  BRK                     \'entry' is the start address
        EQUB 0                  \Your error number
        EQUS "Syntax : <correct syntax>":BRK    \Your error message
.entry  LDA #1:LDY #0:LDX #zp \zp is a zero page location
        JSR osargs              \Get command line
        LDY #0:LDA #0:JSR osargs \Find what filing system
        CMP #5:BNE new_nfs      \It's o.k. if the FS isn't Econet
        LDA #2:LDY #0:JSR osargs \Read version of NFS
                                \(See OSARGS A=2 Y=0)
        CMP #2:BNE new_nfs      \Branch if the NFS is >3.34
.loop1  INY:LDA (zp),Y
        CMP #&D:BEQ error       \If no parameter supplied then error
        CMP #ASC" ":BNE loop1 \Repeat until space is found
.loop2  INY
        LDA (zp),Y
        CMP #ASC" ":BEQ loop2 \Deal with spaces between
                                \ command and parameter
.new_nfs            Your program starts here!
                                \Next letter is at (zp),Y
```

The above program will only work if executed in the I/O processor. If this is to be tube compatible then the block of memory, holding the end of the command line parameter, must be transferred across the tube.

Compatibility between Filing Systems :

Not supported on TAPE or ROM.

Version of NFS

General description

This call returns the version of NFS ROM.

On entry,

A=2
Y=0

On exit,

A=Version of NFS
X,Y preserved
N,V,Z,C undefined

Value in A	Meaning
A=1	ANFS, NFS version 3.40 or greater
A=2	NFS version 3.34 or call not supported

Compatibility between Filing Systems :

Supported on NFS only. If you are trying to write programs which are to be compatible on all filing systems then you must check to see if the current filing system is the network (OSARGS with A=0, Y=0) before issuing this call. For example, if the call is being used to get around the bugs in NFS version 3.34 then first check the filing system, if the filing system is not network then do not use this call, because the result of the call will be A=2 (call not supported).

Read PTR#

OSARGS A=0 Y<>0

General description

Read the sequential pointer (PTR#) leaving the result in four consecutive zero page locations specified in X.

On entry,

A=0
Y=file handle (see OSFIND)
X=zero page location

On exit,

A=0
X,Y preserved
N,V,Z,C undefined

The address pointed to by X (zero page location) holds the sequential pointer in 6502 order as a 4 byte value.

Compatibility between Filing Systems :

Not supported on TAPE or ROM.

Write PTR#

OSARGS A=1 Y<>0

General description

Write the sequential pointer (PTR#) of a file using the value held in four consecutive zero page locations pointed to by X.

On entry,

A=1
Y=file handle (see OSFIND)
X=zero page location

On exit,

A=0 if operation successful, otherwise A=1
X,Y preserved
N,V,Z,C undefined

The contents of the zero page locations are preserved.

Compatibility between Filing Systems :

Not supported on TAPE or ROM.

Read extent

OSARGS A=2 Y<>0

General description

Reads the extent of a file leaving the result in four consecutive zero page locations specified in X.

On entry,

A=2
Y=file handle (see OSFIND)
X=zero page location

On exit,

A=0 if operation successful, otherwise A=2
X,Y preserved
N,V,Z,C undefined

The address pointed to by X (zero page location) holds the extent of the file in 6502 order as a 4 byte value.

Compatibility between Filing Systems :

Not supported on TAPE or ROM.

Write extent

OSARGS A=3 Y<>0

General description

Writes the extent of a file (equivalent to BASIC's EXT#) using the value given in four consecutive zero page locations specified in X.

On entry,

A=3
Y=file handle (see OSFIND)
X=zero page location

On exit,

A=0 if operation successful, otherwise A=3
X,Y preserved
N,V,Z,C undefined

The contents of the zero page locations are preserved.

Compatibility between Filing Systems :

This call is only currently supported on the Advanced Network filing system ROM. However a call of this nature requires cooperation from the file server as well, and not all versions of File Server support it either.

Read space

OSARGS A=4 Y<>0

General description

Reads the amount of space currently allocated to a file.

On entry,

A=4
Y=file handle (see OSFIND)
X=zero page location

On exit,

A=0 if operation successful, otherwise A=4
X,Y preserved
N,V,Z,C undefined

Compatibility between Filing Systems :

This call is only supported on the Advanced Network Filing system ROM.

ANFS Internal Info.

OSARGS A=&80 Y<>0

General description

Return internal information held by the Advanced Network filing system ROM.

On entry,

A=&80
Y=handle to return information on.
X points to four bytes in zero page.

On exit,

The first byte has the handle in file server format.
The second and third byte hold the fileserver number.
The fourth byte holds the following information :-

Bit 0 is the sequence number
Bit 1 is the 'known to be a dir' flag bit
Bit 2 is the 'thought to be URD' bit
Bit 3 is the 'thought to be CSD' bit
Bit 4 is the 'thought to be CSL' bit
Bit 5 is the 'current context' bit
Bit 6 is the EOF error bit, set if the next BGET will fail
Bit 7 is the write flag, set means writable to

Compatibility between Filing Systems :

This call is only supported on the Advanced Network Filing system ROM.

Update file

OSARGS A=&FF

General description

Ensures that any data held temporarily in RAM is transferred to disc. Useful in programs which may run for many hours and do not wish to risk losing data if the power fails etc.

On entry,

A=&FF

On exit,

A=0 if operation successful, otherwise A=&FF
X,Y preserved
N,V,Z,C undefined

Compatibility between Filing Systems :

Not supported on TAPE or ROM. Although this call is supported on the standard version of the NFS ROM the call has no effect because the data will always be up to date on the file server; this is not the case with the ANFS ROM because OSBGET and OSBPUT are buffered internally.

10.6 OSFILE

Call Summary

Entry point &FFDD
Indirected via &212

On entry,

A=Reason code
YX point to the control block

Value in A	Function
A=0	Save a block of memory as a file using the information provided in the parameter block.
A=1	Write the information in the parameter block to the catalogue entry for an existing file (i.e. file name and address).
A=2	Write the load address (only) for an existing file.
A=3	Write the execution address (only) for an existing file.
A=4	Write the attributes (only) for an existing file.
A=5	Read a file's catalogue information, with the file type returned in the accumulator. The information is written to the parameter block.
A=6	Delete the named file.
A=7	Create a file (ANFS only)
A=&FF	Load file to given address.

Save a file

OSFILE A=0

General description

This call saves a block of memory as a file, using the information provided in the control block.

On entry,

A=0
YX point to the control block shown below :-

0	Address of filename
2	Reload address
6	Execution address
10	Start address of data
14	End address of data
18	

On exit,

A,X,Y,C undefined

The control block becomes corrupt.

Compatibility between Filing Systems :

Not supported on read only systems such as ROM filing system.

Change file info.

OSFILE A=1

General description

This call is used to change the reload address, execution address and attributes of a file to the values given.

On entry,

A=1
YX point to the control block shown below :-

0	Address of filename
2	Reload address
6	Execution address
10	Attributes
11	

The attributes of a file are 8 bits corresponding to the following values :-

Bit	Access set on Econet	Meaning if set
7		Undefined on network (locked for other users on some filing systems)
6		Undefined on network (executable by other users on some filing systems)
5	W public access	The file is writeable by other users
4	R public access	The file is readable by other users
3	L	The file is locked
2	M	Undefined on network (executable by you on some filing systems)
1	W owner access	The file is writeable by you
0	R owner access	The file is readable by you

On exit,

A,X,Y,C undefined

Compatibility between Filing Systems :

Not supported on TAPE or read only filing systems such as ROM.

Change reload address

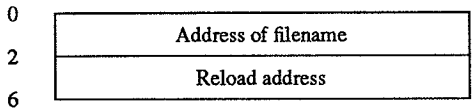
OSFILE A=2

General description

This call is used to change the reload address of a file to that specified in the control block.

On entry,

A=2
YX point to the control block shown below :-



On exit,

A,X,Y,C are undefined

Compatibility between Filing Systems :

Not supported on TAPE or read only filing systems such as ROM.

Change execute address

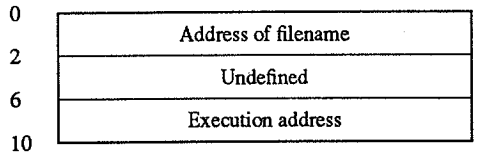
OSFILE A=3

General description

The call is used to change the execution address of a file to that given in the control block.

On entry,

A=3
YX point to the control block shown below :-



On exit,

A,X,Y,C undefined

Compatibility between Filing Systems :

Not supported on TAPE or read only filing systems such as ROM.

Change attributes

OSFILE A=4

General description,

This call is used to change the attributes of a file or directory.

On entry,

A=4
YX point to the control block shown below :-

0	Address of filename
2	Undefined
14	File attributes
15	

See OSFILE A=1 for meaning of file attribute bits.

On exit,

A,X,Y,C undefined

Compatibility between Filing Systems :

Not supported on TAPE or read only filing systems such as ROM.

Read catalogue info.

OSFILE A=5

General description

This call is used to read a file or directory's catalogue information into a control block.

On entry,

A=5
YX point to the control block shown below :-

0	Address of filename
2	undefined
15	

On exit,

X,Y,C undefined
A=0 if object not found, A=1 if object was a file, A=2 if object was a directory.

0	Address of filename
2	Load address
6	Execution address
10	Length
14	Attributes
15	Creation date
17	Undefined
18	

The attributes are 8 bits corresponding to the following values :-

Bit	Access set on Econet	Meaning if set
7	L	Locked for other users
6	R public access	Executable by other users
5	W public access	Writeable by other users
4	R public access	Readable by other users
3	L	Locked for you
2	R owner access	Executable by you
1	W public access	Writeable by you
0	R owner access	Readable by you

Compatibility between Filing Systems :

Not supported on TAPE or ROM.

Delete object

OSFILE A=6

General description

Deletes the filename pointed to by the control block.

On entry,

A=6
YX point to the control block shown below :-

0	Address of filename
2	Undefined
18	

On exit,

X,Y,C undefined
If object not found then A=0 otherwise A>0 and control block updated as OSFILE A=5.

Compatibility between Filing Systems :

Not supported on TAPE or read only filing systems such as ROM.

Create file

OSFILE A=7

General description

This call is the same as the save operation (OSFILE A=0) except that no data is transfered. Consequently the newly created file is filled with zeros.

On entry,

A=0
YX point to the control block shown below :-

0	Address of filename
2	Reload address
6	Execution address
10	Start address
14	End address
18	

On exit,

A,X,Y,C undefined

The control block becomes corrupt.

Compatibility between Filing Systems :

Not supported on read only systems such as ROM filing system, nor on NFS 3.34/3.6, early versions of File Server, early versions of DFS.

Load file

OSFILE A=&FF

General description

This calls loads a file at a specified reload address, or, if the low byte of the execution address parameter is not zero, at the file's own reload address. The control block is updated with the file's catalogue information (load address, execution address, length, attributes).

On entry,

A=&FF
YX point to the control block shown below :-

0	Address of filename
2	Undefined
15	

On exit,

A,X,Y,C undefined

0	Address of filename
2	Reload address
6	Execution address
10	Length
14	Attributes
15	

Compatibility between Filing Systems :

Not supported on TAPE or ROM filing systems.

10.7 OSCLI

Command line

Entry point &FFF7
Indirected via &208

General description

This call is used to send a command line to the Operating system or your currently selected filing system. This is the equivalent of a direct command with a '*' in front of it.

On entry,

A is undefined
YX point to the command line to be interpretated.

On exit,

A,X,Y,C undefined

Example :

```
10 DIM s% 80
20 PROCoscli("CAT")
30 END
40
50DEF PROCoscli($s%)
60 X%=s%:Y%=s% DIV 256
70 CALL &FFF7
80ENDPROC
```

If the procedure is used instead of the keyword OSCLI, then it can be guaranteed that this part of the program will not be sensitive to version 1 of BASIC.

10.8 OSWORD

Call Summary

Entry point &FFF1
Indirects via &20C

On entry,

Value in A	Control byte	Function
A=&10	&81	READ a block of memory from a remote machine
A=&10	&82	WRITE a block of memory to a remote machine
A=&10	&83	CALL location and send argument block to remote machine
A=&10	&84	User procedure call to a remote machine
A=&10	&85	Operating system procedure call to a remote machine
A=&10	&86	HALT a remote machine
A=&10	&87	START a remote machine (after a Halt)
A=&10	&88	Read the machine type and version number of a remote machine
A=&11		Receive block operations
A=&12		Reading arguments
A=&13		Reading/writing station information
A=&14	&00	Communicate with File Server
A=&14	&01	Send Text message
A=&14	&02	Cause remote error

Calls with A=&10 are all varients of *Transmit* see section 10.15 for details of the necessary polling and retry techniques.

10.9 Peek

General description

This returns a block of memory from a remote machine into the local machine. See section 10.15 for information on polling the transmit for success.

On entry,

A=&10
YX point to the address of the control block shown below :-

0	&81
1	0
2	Remote station (station number, network)
4	Pointer to start of local buffer
8	Pointer to end of local buffer
12	Pointer to start of remote machine's buffer
16	

On exit,

A,X,Y undefined

0	Modified
1	Unchanged
16	

Compatibility between Filing Systems :

Only supported by the NFS.

Poke

OSWORD A=&10 (Control byte=&82)

General description

This sends a block of memory from the local machine into the remote machine. See section 10.15 for information on polling the transmit for success.

On entry,

A=&10
YX point to the address of the control block shown below :-

0	&82
1	0
2	Remote station (station number, network)
4	Pointer to start of local buffer
8	Pointer to end of local buffer
12	Pointer to start of remote machine's buffer
16	

On exit,

A,X,Y undefined

0	Modified
1	Unchanged
16	

Compatibility between Filing Systems :

Only supported by the NFS. The NFS need not be the current filing system.

Remote JSR

OSWORD A=&10 (Control byte=&83)

General description

This call sends an argument block to a remote machine, then jumps to a location in the remote machine. See section 10.15 for information on polling the transmit for success.

On entry,

A=&10
YX point to the address of the control block shown below :-

0	&83
1	0
2	Remote station (station number, network)
4	Pointer to start of local buffer for arguments
8	Pointer to end of local buffer for arguments
12	Address to CALL in remote machine
16	

After this call the remote machine is protected against procedure calls and O.S. procedure calls until the parameter block is read. The program in the remote machine must read the parameter block (OSWORD A=&12) before exiting (with a RTS), otherwise the remote machine will remain protected. If the remote machine is a BBC microcomputer then the address to call must be in the I/O processor.

Although interrupts are disabled in the remote machine, they should be enabled if the routine is going to take much longer than 1 mS to complete.

The maximum size of the argument block is 128 bytes.

On exit,

A,X,Y undefined

0	Modified
1	Unchanged
16	

Compatibility between Filing Systems :

Only supported by the NFS. The NFS need not be the current filing system.

Remote procedure

OSWORD A=&10 (Control byte=&84)

General description

This passes a block of memory to a remote machine and causes an event (number 8) in that machine. The program in the remote machine must intercept the event number (procedure number held in YX) and read the argument block (OSWORD A=&12) before exiting with an RTS.

Note that the argument block must be read even if there are no arguments, because the Rx control block will not be reopened until this has happened.
Machines with Econet version 3.34 may crash, because stations greater than 240 can override the machine protection, therefore they can overwrite the argument block before it is read. See section 10.15 for information on polling the transmit for success.

On entry,

A=&10
YX point to the address of the control block shown below :-

0	&84
1	0
2	Remote station (station number, network)
4	Pointer to start of local buffer for arguments
8	Pointer to end of local buffer for arguments
12	Procedure number
14	

In the remote machine the Accumulator (A register) holds the event number (which will be 8). X will hold the low byte of the procedure number and Y will hold the high byte.

On exit,

A,X,Y undefined

0	Modified
1	Unchanged
14	

Example

The first program uses the user procedure call to cause an event in a remote machine (held in the variable stn% : line 60). Note that the second program MUST be running in the remote station, otherwise the remote station will become protected.

```
 10 REM Program to send Events to a remote machine
 20 DIM blk% 100,buffer% 100
 30 osword=&FFF1:osbyte=&FFF4
 40
 50 REPEAT
 60 stn%=4 : cntr%=100
 70 REPEAT
 80 blk%?0=&84 : blk%?1=0 : blk%!2=stn%
 90 blk%!4=buffer% : blk%!8=buffer%+4
100 blk%!12=2 :REM Procedure number
110 X%=blk% : Y%=blk% DIV 256 : A%=&10
120 CALL osword
130 cntr%=cntr%-1
140 UNTILFNpoll_transmit OR cntr%=0
150
160 UNTIL cntr%=0
170 PRINT"I can't send an event to machine ";stn%
180 END
190
200DEF FNpoll_transmit
210 LOCAL A%:A%=&32
220 REPEAT V%=(USR(osbyte) AND &FF00)
230  UNTIL (V% AND &8000)=0
240=((V% AND &7F00)=0)
```

This program is to be run in the remote machine. When an event occurs, the event routine increments a location on the screen.

```
 10 REM  Detecting an event sent through the Econet
 20 REM    (C) A.J.Engeham, SJ Research
 30 REM _____________________________________
 40 DIM code &100,buffer 100
 50 osword=&FFF1:osbyte=&FFF4
 60 MODE 7
 70 PROCassemble
 80 CALL set_up_event
 90 PRINT"The event routine has been set up"'
100 PRINT" - waiting for event to happen"
110 END
120
130DEF PROCassemble
140 FOR pass=0 TO 2 STEP 2
150 EVENTV=&220:zp=&70
160 P%=code
170[OPT pass
180.set_up_event
190 LDA EVENTV:STA zp            \Save old event handling routine
200 LDA EVENTV+1:STA zp+1
210 LDA #handle_eco MOD 256
```

```
  220 SEI                           \Changing Event vector interupts
must
  230                               \ be disabled.
  240 STA EVENTV
  250 LDA #handle_eco DIV 256:STA EVENTV+1
  260 CLI                           \Vector done reenable interupts
  270 LDA #14:LDX #8:JSR osbyte     \Enable Econet events
  280 RTS
  290
  300\******** Event handling routine *********
  310.handle_eco
  320 PHP
  330 CMP #8:BNE not_econet_event   \If not 8 then the event was not
caused
  340                               \ by the Econet
  350 PHA:TXA:PHA:TYA:PHA           \It was the Econet so save registers
  360
  370 CPX #2                        \Check for User procedure call 2
  380 BNE dud_eco_call              \Not the procedure call sent from
  390 CPY #0:BNE dud_eco_call       \ my machine
  400 INC &7E00                     \Do something exciting
  410.dud_eco_call
  420 LDX #buffer MOD 256:LDY #buffer DIV 256 \YX points to the buffer
  430 LDA #&12:JSR osword           \Read arguments
  440 PLA:TAY:PLA:TAX:PLA           \Restore registers
  450.not_econet_event
  460 PLP                           \Restore flag
  470 JMP (zp)                      \Pass event back to OS to handle
  480 ]
  490 NEXT
  500ENDPROC
```

Compatibility between Filing Systems :

Only supported by the NFS. The NFS need not be the current filing system.

Remote Insert key

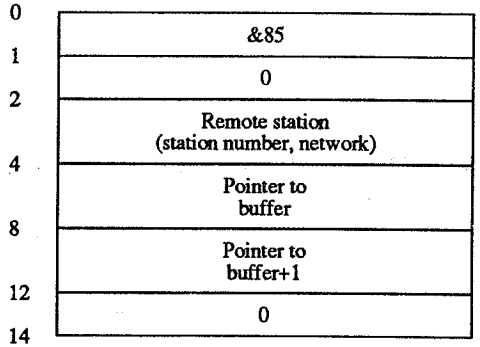
OSWORD A=&10 (Control byte=&85)

General description

This call inserts a byte into a remote machine's keyboard buffer and is used by the 'send a line of text' command (OSWORD A=&14 Control byte=1), which should usually be used in preference to this call. See section 10.15 for information on polling the transmit for success.

On entry,

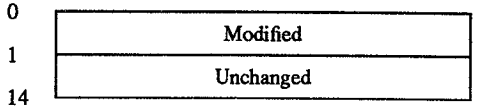
A=&10
YX point to the address of the control block shown below :-



The buffer (which is 1 byte long) holds the byte to send to the remote station.

On exit,

A,X,Y undefined



Example :

```
 10 REM Insert a character into a remote machine's keyboard buffer
 20 REM  (C) A.J.Engeham, SJ Research
 30 REM
 40
 50 DIM blk% 16,buffer% 1
 60 osword=&FFF1:osbyte=&FFF4
 70 INPUT"Station number : ";stn%
 80 PRINT"Character to send : ";
 90 ?buffer%=GET
100 tries%=20
```

```
110 REPEAT
120  blk%?0=&85
130  blk%?1=0
140  blk%!2=stn%
150  blk%!4=buffer%
160  blk%!8=buffer%+1
170  blk%!12=0
180  X%=blk%:Y%=X% DIV 256
190  A%=&10:CALL osword
200  tries%=tries%-1
210  REPEAT
220   A%=&32
230   UNTIL (USRosbyte AND &8000)=0
240  tries%=tries%-1
250  UNTIL (USRosbyte AND &7F00)=0 OR tries%=0
260 IF tries%=0 PRINT"Unable to send character":END
270 PRINT"Character sent"
280 END
```

Compatibility between Filing Systems :

Only supported by the NFS. The NFS need not be the current filing system.

Start REMOTE

OSWORD A=&10 (Control byte=&85)

General description

This call is used to start a REMOTE. See section 10.15 for information on polling the transmit for success.

On entry,

A=&10
YX point to the address of the control block shown below :-

0	&85
1	0
2	Remote station (station number, network)
4	Pointer to buffer
8	Pointer to buffer+1
12	1
14	

On exit,

A,X,Y undefined

0	Modified
1	Unchanged
14	

Compatibility between Filing Systems :

Only supported by the NFS. The NFS need not be the current filing system.

Update workspace

OSWORD A=&10 (Control byte=&85)

General description

This call causes a remote machine to write the state of its screen to a position in its Econet workspace (whence it can be read by the local machine by PEEKing this workspace). See section 10.15 for information on polling the transmit for success.

On entry,

A=&10
YX point to the address of the control block shown below :-

0	&85
1	0
2	Remote station (station number, network)
4	Pointer to buffer
8	Pointer to buffer+1
12	2
14	

On exit,

A,X,Y undefined

0	Modified
1	Unchanged
14	

In the remote machine the data is written to the address (in the I/O processor) pointed to by locations &9E and &9F plus &E9. This data is shown below :-

0	Address of top of screen
2	Palette (physical colours defined on screen)
18	Mode number (0-7)
19	Address of start of screen
21	Mark space for colours
23	

Example :

```
 10 REM  Info. about a remote m/c : Demonstration for OS procedure 2
 20 REM    (C) A.J.Engeham, SJ Research
 30 REM _____________________________________________________________
 40
 50 DIM blk% 100,buffer% 100
 60 INPUT"Station number ",stn%
 70 osword=&FFF1:osbyte=&FFF4
 80
 90 PROCtransmit(&85,1,2)
100 PROCtransmit(&81,2,&FFFF009E)
110 PROCtransmit(&81,30,&FFFF0000+(buffer%!0 AND &FFFF)+&E9)
120
130 MODE 7
140 PRINT"Machine number ";stn%
150 PRINT" MODE ";buffer%?18
160 PRINT" Address of start of screen=&";~buffer%!19 AND &FFFF
170 PRINT" Address of top of screen=&";~buffer%!0 AND &FFFF
180 PRINT'"Palette"'"_______"'"Logical  Physical"
190 FOR I%=0 TO 15
200 PRINTTAB(3);I%TAB(12);I%?(buffer%+2)
210 NEXT
220END
230
240DEF FNpoll_transmit
250 LOCAL A%:A%=&32
260 REPEAT V%=(USR(osbyte) AND &FF00)
270 UNTIL (V% AND &8000)=0
280=((V% AND &7F00)=0)
290
300DEF PROCtransmit(cb%,no_of_bytes%,start%)
310 LOCAL cntr%,X%,Y%,A%
320 cntr%=20
330 REPEAT
340  blk%!0=cb%
350  blk%!2=stn%
360  blk%!4=buffer%
370  blk%!8=buffer%+no_of_bytes%
380  blk%!12=start%
390  X%=blk%:Y%=blk% DIV 256
400  A%=&10:CALL osword
410  cntr%=cntr%-1
420  UNTIL FNpoll_transmit OR cntr%=0
430 IF cntr%=0 THENPRINT"Machine not listening":END
440ENDPROC
```

Compatibility between Filing Systems :

Only supported by the NFS. The NFS need not be the current filing system.

Fatal Error

OSWORD A=&10 (Control byte=&85)

General description

This causes a fatal error in a remote machine.
It is easier to use the high level fatal error call (OSWORD A=&14 Control byte=2). See section 10.15 for information on polling the transmit for success.

On entry,

A=&10
YX point to the address of the control block shown below :-

0	&85
1	0
2	Remote station (station number, network)
4	Pointer to buffer
8	Pointer to buffer+1
12	3
14	

On exit,

A,X,Y undefined

0	Modified
1	Unchanged
14	

Compatibility between Filing Systems :

Only supported by the NFS. The NFS need not be the current filing system.

Halt

OSWORD A=&10 (Control byte=&86)

General description

Halts all non-interrupt operations in the I/O processor of a remote machine. If a tube is running on the remote machine, then that will continue running until it tries to comunicate with the I/O processor. See section 10.15 for information on polling the transmit for success.

On entry,

A=&10
YX point to the address of the control block shown below :-

0	&86
1	0
2	Remote station (station number, network)
4	

On exit,

A,X,Y undefined

0	Modified
1	Unchanged
14	

Compatibility between Filing Systems :

Only supported by the NFS. The NFS need not be the current filing system.

Continue

OSWORD A=&10 (Control byte=&87)

General description

Restarts the I/O processor of a remote machine after a Halt command (OSWORD A=&10 Control byte=&86). See section 10.15 for information on polling the transmit for success.

0	&87
1	0
2	Remote station (station number, network)
4	

On exit,

A,X,Y undefined

0	Modified
1	Unchanged
14	

Compatibility between Filing Systems :

Only supported by the NFS. The NFS need not be the current filing system.

Identify Machine

OSWORD A=&10 (Control byte=&88)

General description

This call interrogates a remote machine returning codes containing values to distinguish between manufacturers, machine types and software versions. See section 10.15 for information on polling the transmit for success.

On entry,

A=&10
YX point to the address of the control block shown below :-

0	&88
1	0
2	Remote station (station number, network)
4	Pointer to start of local buffer
8	Pointer to end of local buffer (start+4)
12	

On exit,

A,X,Y undefined

0	Modified
1	Unchanged
14	

The first four bytes of the buffer are relevant. The meaning of these bytes are shown below :-

Byte 1

This is defined by manufacturers and is intended to indicate the hardware design of the machine. The following are currently defined :-

Value	**Type of machine**
SJ Research	
FF	Z80 CP/M
FE	SJ Research File Server
FD	RM 480Z
FC	Nascom 2 (running CP/M)
FB	IBM Interface board

FA	SCSI Interface board

Acorn

01	BBC microcomputer (includes B+)
02	Atom
03	System 3 or System 4
04	System 5
05	Master
06	Electron
07	Reserved for future machine
08	Reserved for future machine
09	Communicator
0A	Econet Terminal
0B	File Store
0C	Compact

Byte 2

This byte indicates the manufacturer. The following are currently defined :-

Value	Manufacturer
00	Acorn
01	Torch
02	Reuters
FF	SJ Research

Bytes 3 & 4

These contain the low and high bytes of the software release version (in base 16) and are manufacturer specific.

Compatibility between Filing Systems :

Only supported by the NFS. The NFS need not be the current filing system.

10.10 Set up Rx block

OSWORD A=&11

General description

This call is used to set up a receive block in your local machine. Once this has been done, data which is transmitted, or broadcast, on your port number, can be received.

On entry

A=&11
YX point to the address of the control block shown below :-

0	0
1	&7F
2	Port number
3	Station number
5	Pointer to start of buffer
9	Pointer to end of buffer
13	

If the port number is zero then the receive block will receive data transmitted on any port number. If the station number is zero then the receive block will receive data sent from any station, as long as the port number is correct. Note that the size of the buffer must be big enough to receive all the data transmitted otherwise the data will be rejected, and the receive block left open. To select the correct port number for your application see the Econet Standards Group paper 0001 at the end of this section.

On exit,

A,X,Y undefined

0	Receive block number
1	Unchanged
13	

If the receive block number allocated is zero then all the available receive blocks have been used. To correct this delete an unused receive block and retry the call.

Read Rx block

OSWORD A=&11

General description

Once data has been received into an open receive block a read operation is neede to discover the source of the data and the number of bytes actually received. This call allows a copy of the updated control block to be written to the address pointed to by YX. Reading a receive block deletes the old receive block and allows that receive block to be reallocated. If the receive block was open to all stations, by using a station number of 0, then the actual machine number that sent the packet will be written; this is also the case for the port number. See section 10.15 for details of polling to see when data has arrived.

On entry,

A=&11
YX point to the control block shown below :-

0	Control block number
1	Undefined
13	

On exit,

A,X,Y undefined

0	Control block number
1	Control byte of transmit packet from remote machine
2	Port number of remote machine
3	Station number of remote machine
5	Pointer to start of data
9	Pointer to end of data
13	

10.11 Read arguments

OSWORD A=&12

This call allows arguments to be read after a remote JSR (OSWORD A=&10 control byte=&83) and a user procedure call (OSWORD A=&10 control byte=&84). The remote machine will have the protect status bits set until this call is used, this is to prevent the arguments from being overwritten. The maximum size of the argument block is returned by OSWORD A=&13, function code=9.

On entry,

A=&12
YX point to the address in memory to put the arguments, of which the first two bytes are the source station number.

On exit,

A,X,Y undefined.

10.12 OSWORD A=&13 Call Summary

General description

These calls allow general local station information to be read and written.

On entry,

A=&13
YX point to the control block with the first byte determining the function and number of parameters. Except for function codes 0,1,6 and 7 the NFS does not need to be selected to perform the operations.

Control byte	Function
0	Read File Server number
1	Write File Server number
2	Read Printer Server number
3	Write Printer Server number
4	Read Protection mask
5	Write Protection mask
6	Read context handles
7	Write context handles
8	Read local station number
9	Read number, and size, of arguments
10	Read error number

On exit,

A,X,Y undefined
Control block modified

FS number

General description

These calls allow the File Server number to be read and written. This value is used for all File Server commands including *I AM <user identifier> and OSWORD A=&14. To change onto a different File Server, context handles need to be written otherwise it is likely that any further operations will return the error 'Channel'. The ANFS utility '*FS' does both of these operations.

The BBC NFS ROM defaults to the value 0.254 (i.e. station 254 on the local network). On the Master, Compact and Econet Terminal the default File Server number is held in CMOS ROM and is configured by the command '*CONFIGURE FS <new number>'.

These calls should only be attempted with Econet selected as the current filing system.

On entry

A=&13
YX point to the control block shown below :-

On exit,

A,X,Y undefined
For read operation the control block is modified as shown above. For write operation the control block is undefined.

PS number

OSWORD A=&13 (control byte=2/3)

General description

These calls concern the machine using a Print Server. To send a listing it is also necessary to set the printer type to the network, by issuing the command '*FX5,4'. The BBC NFS ROM defaults to the value 0.235 (i.e. station 235 on the local network). On the Master, Compact and Econet Terminal the default File Server number is held in CMOS ROM and is configured be the command '*CONFIGURE PS <new number>'.

On entry,

A=&13
YX point to the control block shown below :-

0	Control byte 2 = Read PS number 3 = Write PS number
1	Print Server local number
2	Print Server network number
3	

On exit,

A,X,Y undefined
For read operation the control block is modified as shown above. For write operation the control block is undefined.

Protection mask

OSWORD A=&13 (control byte=4/5)

General description

These calls allow the user to read and write the protection mask of the local machine. Note that if a remote machine attempts to do an operation requiring parameters returned then various protection bits will be set until the parameters have been read (using OSWORD A=&12).

On entry,

A=&13
YX point to the control block shown below :-

0	Control byte 4 = Read protection mask 5 = Write protection mask
1	Value of protection mask
2	

If a bit of the protection mask is clear then the remote operation is allowed. The possible values of the protection mask are shown below :-

bit	Operation
0	Peek
1	Poke
2	JSR
3	User procedure call
4	Operation system procedure call
5	Halt

On exit,

A,X,Y undefined
For read operation the control block is modified as shown above. For write operation the control block is undefined.

Context handles

OSWORD A=&13 (control byte=6/7)

General description

Three directory handles describing the current environment are required for most File Server operations. These are returned by the File Server by the command '*I AM <user identifier>'. The three handles are :-

User root directory (URD) - starting directory for that user. If the command '*DIR' is issued then the user will be returned to this directory.

Currently selected directory (CSD) - the directory that the user is currently in.

Library directory (LIB) - the directory that is searched if a filename is not found in the CSD.

These calls should only be attempted with Econet selected as the current filing system.

On entry,

A=&13
YX point to the control block shown below :-

0	Control byte 6 = Read context handles 7 = Write context handles
1	URD
2	CSD
3	LIB
4	

On exit,

A,X,Y undefined
For read operation the control block is modified as shown above. For write operation the control block is undefined.

Read station

OSWORD A=&13 (control byte=8)

General description

This call returns the local station number.

On entry,

A=&13
YX point to the control block shown below :-

0	8=Read local station number
1	undefined
2	

On exit,

A,X,Y undefined
Control block is modified to hold local station number.

Example :

```
 10 DIM blk% 2
 20 PRINT FNstation
 30 END
 40
 50DEF FNstation
 60 A%=&13
 70 X%=blk%:Y%=blk% DIV 256
 80 blk%?0=8
 90 CALL &FFF1
100=blk%?1
```

Read arg. info.

OSWORD A=&13 (control byte=9)

General description

This call is used to find the size of the argument block used by remote JSR and remote procedure call, currently held by the NFS, and the maximum size of argument block possible.

On entry,

A=&13
YX point to the control block shown below :-

0	9=Read arguments
1	undefined
3	

On exit,

A,X,Y undefined
Control block is modified as shown below :-

0	undefined
1	Number of arguments
2	Maximum number of arguments
3	

Read error

OSWORD A=&13 (control byte=10)

General description

This call is used to find the actual error number after a 'catch-all' error (error number &A8) has happened, or the screen MODE after a 'Mode x' error from '*VIEW'.

On entry,

A=&13
YX point to the control block shown below :-

0	10=Read errors number
1	undefined
2	

On exit,

A,X,Y undefined
Control block is modified as shown below :-

0	undefined
1	Error number
2	

10.13 Send to FS

OSWORD A=&14 (control byte=0)

General description

This call is used to communicate directly with the File Server and is detailed fully in the section detailing the File Server interface. The network must be the current filing system.

On entry,

A=&14
YX point to the control block shown below :-

0	0=Communicate with File Server
1	Size of whole of block (x)
2	0 - NFS will put reply port here
3	FS Function code
4	0 - NFS will put URD handle here
5	0 - NFS will put CSD handle here
6	0 - NFS will put LIB handle here
7	Rest of FS transmit block
x	

On exit,

A,X,Y undefined
Control block is modified as shown below :-

0	0
1	Size of rest of block
2	Command code
3	Return code
4	Rest of file server receive block
x	

Send text

OSWORD A=&14 (control byte=1)

General description

This call sends a message to a remote machine and is used by the library utility *NOTIFY. The NFS must be the current filing system.

On entry,

A=&14
YX point to the control block shown below :-

0	1=Send text
1	Destination station
3	Text, terminated by 0 or &0D
x	

On exit,

A,X,Y undefined
Contents of control block is undefined.

The maximum size of text is 128 bytes when performed from a 2nd processor and 250 bytes when performed from the I/O processor. This call should not be used from within an interrupt routine.

Cause remote error

OSWORD A=&14 (control byte=2)

General description

This call causes a fatal error in a remote machine, by executing a BRK instruction followed by another 0, thus generating error 0. The NFS must be the current filing system. This is only guaranteed to be a fatal error in BASIC 2 and above. In BASIC 1 the ON ERROR will still trap fatal errors. Some other languages also incorrectly implement trapping of error 0.

On entry,

A=&14
YX point to the control block shown below :-

0	2=Cause fatal error
1	Destination station
3	

On exit,

A,X,Y undefined
Contents of control block is undefined.

10.14 OSBYTE

Entry point &FFF4
Indirected via &20A

On entry,

A=Function code

Value in A	Function
A=&32	Poll transmit block
A=&33	Poll receive block
A=&34	Delete a receive block
A=&35	Sever remote connection

On exit,

X=Result dependent on the function
A,Y,C undefined

Call Summary

Poll Transmit

OSBYTE A=&32

General description

This call is used to poll a transmit operation for completion. If an error occured the error code will be returned in X.

On entry,

A=&32

On exit,

X=0 if the transmission has been completed successfully, otherwise the following bits in X are relevant. Bit 7 is the top bit, bit 0 is the bottom bit.

bit	state	Meaning
7	0	completed
	1	in progress
6	0	successful
	1	failed
5	0	
4	0	
3	0	
2-0		hold the error code

Error codes

0	Line jammed.
1	Incomplete or bad four-way handshake
2	No acknowledge to scout packet
3	No clock
4	Bad transmit control block

A,Y,C undefined

Example :

```
DEF FNpoll_Tx
  LOCAL Result%,A%
  A%=&32
  REPEAT
    Result%=(USR(osbyte) AND &FF00) DIV &100)
  UNTIL Result%<&80
=Result%
```

The BASIC function above will poll the last transmit operation until it has been completed. When the function returns, if the result is non zero, then there has been an error in the transmission and the transmit command may have to be redone.

Poll Receive

OSBYTE A=&33

General description

This call is used to check whether anything has been received in a receive block.

On entry,

A=&33
X=control block number

On exit,

X has the top bit set if a message has been received.
A,Y undefined
C undefined

Example :

```
DEF FNpoll_Rx(X%)
  LOCAL Result%,A%
  A%=&33
=((USRosbyte AND &8000)<>0)
```

The above example, when given a control block number as a parameter, will return -1 (TRUE) if anything has been received.

```
10 CB%=FNset_up_Rx      :REM Your function which sets up a receive block
20                      :REM  and returns the control block number.
30 REPEAT
40 UNTIL FNpoll_Rx(CB%)
50 PRINT"Something has been received!"
```

Delete receive block

OSBYTE A=&34

General description

This call is used to delete a receive block.

On entry,

A=&34
X=control block number

On exit,

X control block number.
A,Y undefined
C undefined

10.15 Transmitting

To transmit a packet over the network using the BBC microcomputer it is necessary to set up a *control block* in memory. A transmit control block will always take the form :-

0	Control byte (top bit set)
1	Port (must be zero for immediate operations)
2	Station number
4	Start of data
8	End of data
12	Remote start of data
16	

The *control byte* always has the top bit set. The rest of the byte is called a *control code* which, with the exception of immediate operations, makes no difference what value is chosen.

Check that the port number is within the correct range for your application by checking it against the Econet Standards Group paper 0001.

General procedure for transmitting

1) Set up the control block and the number of retries to perform.

2) Store the control byte in the control block.

3) Set the accumulator to &10 and YX to point to the control block.

4) CALL OSWORD to perform the call.

5) Read the control byte of the control block. If this is zero then the transmission has failed to start so go to step 2.

6) Poll for completion (Using OSBYTE A=&32).

7) If not completed (i.e. top bit of X register still set) go to step 6.

8) Has transmission worked (Using OSBYTE A=&32 to find whether X is 0).

9) If there was a non-fatal error in the transmission then decrement the number of retries and if the number of retries is greater than 0 go to step 2.

10) If the error was fatal or the number of retries=0 then finish otherwise return.

The terms *fatal* and *non-fatal* refer to the type of error returned by OSBYTE with A=&32. The errors 'line

jammed' (&40), 'no clock' (&43) and 'bad transmit control block' (&44) are all fatal: this is because they all need manual intervention by the user to cure them, a detailed explaination of these errors is given in appendix A.

The errors 'four way handshake damaged' (&41) and 'no scout acknowlegement' (&42) are termed non-fatal. The error 'no scout acknowlegement' is normally caused by either, the remote machine having no receive block open to receive the data, the machine has the protection status set or the machine is not on the network. The error 'four way handshake damaged' is normally caused by the receive buffer size of the remote machine not being big enough, the data colliding with someone else transmitting or electrical interference. It should be noted that the Acorn BBC B+ and early Master series microcomputers do not have any collision detect hardware (although it can be fitted) and so it is possible that it may not always transmit correctly, especially on the broadcast operation.

Broadcasting

This is a special version of the transmit operation. It allows a station to simultaneously send eight bytes of data to every other station on every network, that has a receive block open and is currently the only method of communicating with the *bridge*. However, the call has no handshaking so it is not possible to guarantee that the remote stations received the data. The control block is shown below :-

0	Control byte (top bit set)
1	Port
2	&FFFF
4	Data
12	

If the remote station has a receive block open on the correct port number then the data will be received like a normal transmit, except that if the remote station has NFS version 3.34 the data will be written over the pointers of the receive control block instead of the location pointed to.

Transmitting under interrupts

If an application requires performing a transmission under interrupts then there is a special precaution to take. This is because the transmit routine interrupts and so there is the possibility that a transmission under interrupts may occur whilst a transmission is already taking place. Thus polling the transmit will get the result of the last transmission. To get around this problem the following *fix* must be done :-

Before transmitting save the old transmission poll flag by :-

```
loop
  LDY #&6F
  LDA (&9C),Y
  BMI loop
  PHA
```

Having successfully transmitted, restore the old transmission poll flag by :-

```
  LDY #&6F
  PLA
  STA (&9C),Y
```

It is essential that the above routines are executed in the I/O processor.

10.16 Receiving

A receive block is needed to receive a transmission from a remote machine. The only exceptions to this are the immediate operations, these are handled specially by the NFS ROM.

Before a receive block can be used it must be set up. This is performed by setting up a control block and CALLing OSWORD A=&11. If a valid receive block number is returned then the receive block is open. The receive block will only be filled if all the following conditions are met :-

1) The correct station replies
2) The data is sent on the correct port number
3) The amount of data sent is not bigger than the buffer size reserved.

10.17 Port numbers

The information below is a copy of a paper by the Econet Standards Group.

ESG paper 0001

This standard defines the way in which port number may legally be used. They should only be used for the purposes given in the following list.

00	Illegal (note 1)
01-0F	Reply only (note 2)
10-2F	Torch Computers (note 3)
30-8F	Reserved for future allocation
90-9F	Acorn Computers (note 3)
A0-AF	SJ Research (note 3)
B0-CF	Reserved for future allocation
D0-DF	Acorn Computers (note 3)
E0-FE	User Programs
FF	Illegal

Notes

1) Used to provide immediate operations.

2) These ports may be used by anyone for any purpose, subject to certain restrictions. They may not be used to initiate a new protocol, but may be used to receive replies within a protocol initiated using some other port. When receiving data on these ports, the receive block must be open for a specific station, and not for all stations. Data should never be sent on these ports unless it is certain that the recipient is prepared for that data, and is carrying out the same protocol. Broadcasts may not be sent on these ports.

These ports must only be used by software which has complete control of one end of the connection, and can ensure that no other software will attempt to transmit or receive on these ports at the same time. It is suggested that unsolicited traffic on these ports be discarded immediately by any software that receives it.

3) Where numbers are allocated to a particular organisation, that organisation will control the use of those numbers. However, it is to be hoped that, as new standards are agreed, some or all of these numbers will be allocated to specific puproses at a later date.

Currently used ports

Ports which are currently used are :-

99	Command port; used for sending a command to the file server.
9E	Printer server status reply port; used by the printer server to reply to a client communicating on port 9F.
9F	Printer server status port; used to ask for current status from the printer server.
A0	Used by the SJ Research communication protocol 'FAST'.
D1	Printer server data port; used for communicating to the print server.

10.18 The Bridge

The number of stations on any one network is limited to 254, because the station number is a one byte number and values 255 and 0 are reserved for special purposes. An Acorn bridge is used to connect two Econet networks together; bridges can be used to connect up to 127 networks together, giving a maximum of 32258 stations in total.

Another use of a bridge is to split a long network so that the data transfer rate inside a network can be fast. This is because each network will be shorter; only when data is being transferred between networks does the data transfer rate slow down. If a fault appears on one of the networks then the other network will continue to work; this is particularly useful for developing and testing new Econet hardware.

The last use of a bridge is to allow otherwise illegal network layouts, the most common of these being a 'T' junction.

How to use the bridge

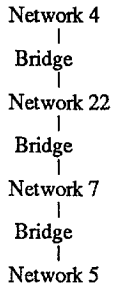
All the primitives supplied by the NFS ROM have two bytes reserved for the station number. The first byte is the station number and the second byte is the network number. To communicate with machines on the local network the network number should be 0.

For a station on network 4 to communicate with another station on network 4, the network number would be 0. But, for a station on network 5 to communicate with a station on network 4, the network number would be 4.

When the bridge is not transferring data, it looks at all the scout packets being sent on both of the networks that it is bridging across, called the networks *adjacent* to the bridge. If the scout packet contains a network number which is within reach through the bridge, then the bridge will perform a four-way handshake between the networks to transfer the data.

To do a four way handshake the bridge receives data from one adjacent network and puts it into its own RAM. While this is done, the bridge sends *flags* on the other adjacent line to prevent any network activity. Once the transfer of data has been completed, the bridge moves to flagging the first network instead, while it transfers the data from its internal RAM onto the second network. Because the bridge has to wait until it has received all the data before re-transmitting it, the transfer rate halves when data is transferred across a bridge. Note that data being transferred between networks 4 and 5, in the system shown overleaf, would be approximately one quarter of the speed of a transfer inside network 4 (the actual figure depends on the clock speeds for each network).

When a bridge is switched on, it only knows about its adjacent networks. So it *broadcasts* a reset packet to find out what other networks are available. The other bridges on the network all update their network lists.



The bridge can be interrogated to find the number of the local networks, version number of the software running in the bridge and the numbers of any other bridges on the network. In addition to these calls the bridge also responds to a reset message and a normal data transfer message.

As the bridge does not use immediate operations, for transmissions, it is necessary to set up a receive block, to receive status information. Thus the process for communicating with the bridge is :-

1) Set up a receive block.
2) Broadcast the data to bridge.
3) Poll receive block until packet received or a timeout occurs.
4) If the polling time was excessive delete the receive block and finish.
5) Read the receive block and return results.

Normally the user will never need to communicate with the bridge as all necessary communications are handled by the network filing system in the computer. ANFS asks the bridge which network number it is on when the machine is first powered on. This is so that the user can type *I AM 1.254 USER when he is on network 1. ANFS knows that the current network number is 1 so before sending the packet it changes the network number to 0.

There are two useful messages that can be sent to the bridge. These are the 'what net' and the 'is net' calls. The 'what net' asks the bridge what the number of the local network is. The packet to broadcast is :-

0	&82
1	&9C
2	&FFFF
4	"BRIDGE",reply port,0
12	

The remote bridge's reply packet will be 2 bytes long containing the local network number and the version of the software running in the bridge.

The other useful message that can be sent to a bridge is the 'is net' packet. This call is used to ask the bridge if it knows about network N, the bridge will perform a four-way handshake if the bridge number is

```
460 LOCAL redo
470 redo=5
480
490 REPEAT
500  tx_blk%?0=&83
510  tx_blk%?1=&9C  :REM Broadcast on port &9C
520  tx_blk%!2=&FFFF :REM Broadcast operation
530  $(tx_blk%+4)="BRIDGE"
540  tx_blk%?10=reply_port
550  tx_blk%?11=network%
560  X%=tx_blk%:Y%=tx_blk% DIV 256
570  A%=&10:CALL osword
580  A%=&32
590   REM Wait for completion
600  X%=FNpoll_tx
610  redo=redo-1
620 UNTIL redo=0 OR X%=&40 OR X%=&43 OR X%=&44 OR X%=0
590   REM Wait for completion
600  X%=FNpoll_tx
610  redo=redo-1
620 UNTIL redo=0 OR X%=&40 OR X%=&43 OR X%=&44 OR X%=0
630 IF X%>0 THEN PRINT"Broadcast failed, error &";~X%:PROCdelete:END
640ENDPROC
650
660DEF PROCdelete
670 REM delete a receive block
680 A%=&34
690 X%=blk%?0
700 CALL osbyte
710ENDPROC
720
730DEF FNpoll_tx
740 REPEAT UNTIL NOT ((USRosbyte) AND &8000)
750=(((USRosbyte) AND &FF00) DIV &100)
760
770DEF PROCdisplay(no%)
780 PRINT"Found network ";no%
790 PROCdelete
800 PROCset_up_rx
810ENDPROC
```

10.19 Printers

The protocol used by SJ Research to locate named printer resources is an adaptation of the protocol used by Acorn (and implemented in ANFS). The extra complexity is introduced by the possibility of having multiple printers at the same station, such that the printer server has to keep track of the current printer selection for each user. The printer status protocol is used : a status request is broadcast, containing the name of the printer desired, and the station, making the broadcast can chose which of the responding servers to select.

Print Server Status Enquiry Protocol

Client to Print Server

Port	&9F
Data-Data+5	6 character printer name padded with spaces
Date+6-Data+7	Two byte reason code (=1 for print status enquiry, =6 for printer name)

Print Server to Client

Port	&9E
Data	1 byte status report
Data+1-Data+2	Two byte station number of machine with which the server is busy

Bits 0-2 of the status byte give the status of the client's input to the printer via the network. Bits 3-4 give the status of the output from the print server to the printer. Bits 5-7 are reserved for future use and currently return zero. Currently defined status values are :-

Input

0-Ready
1-Busy
2-Jammed (general software problem)
3-Jammed, due to printer offline (general hardware problem)
4-Jammed, due to disc full, directory full or similar
5-User not authorised to use printer
6-Spooler going offline / operator has barred input
7-Reserved

Output

0-ready
1-Printer offline
2-Printer jammed (ie has not accepted data for a long time)

Typical responses include :

00000-Normal OK status
01000-Printer offline, but spooler still accepting data
00100-Printer OK, but disc temporarily full
01100-Printer offline, disc full

Print drivers should poll the printer as usual and proceed only if (status AND 7)=0

held in its internal table otherwise the bridge will not reply.

The broadcast packet is :-

0	&83
1	&9C
2	&FFFF
4	"BRIDGE",reply port,N
12	

The most obvious use of the above call is to find out the numbers of all other networks available. The BASIC program below does just this :-

```
 10 REM Find out what other networks are available
 20 REM    (C) 1986, A.J.Engeham, SJ Research
 30 REM _____________________________________________
 40
 50 DIM blk% 100,tx_blk% 20,buffer% 100
 60
 70 osword=&FFF1:osbyte=&FFF4
 80 reply_port=&8A:retries=10
 90 PROCset_up_rx
100
110 FOR bridge%=1 TO 127
120 FOR dummy%=1 TO 3
130 PROCbroadcast(bridge%)
140 NEXT
150
160 dummy%=0
170 REPEAT
180 dummy%=dummy%+1
190 UNTIL dummy%=retries OR FNpoll_rx
200
210 IF FNpoll_rx THENPROCdisplay(bridge%)
220 NEXT
230 PROCdelete
240 PRINT"Finished"
250END
260
270DEF PROCset_up_rx
280 blk%?0=0  :REM control block number put here
290 blk%?1=&7F
300 blk%?2=reply_port
310 blk%!3=&0  :REM Receive from any stations
320 blk%!5=buffer%
330 blk%!9=buffer%+8
340 X%=blk%:Y%=blk% DIV 256
350 A%=&11
360 CALL osword
370 IF blk%?0=0 THENPRINT"Can't open a Rx block":END
380ENDPROC
390
400DEF FNpoll_rx
410 A%=&33
420 X%=blk%?0    :REM Get the rx block number
430=((USRosbyte) AND &8000)<>0
440
450DEF PROCbroadcast(network%)
460 LOCAL redo
470 redo=5
480
490 REPEAT
500  tx_blk%?0=&83
510  tx_blk%?1=&9C  :REM Broadcast on port &9C
520  tx_blk%!2=&FFFF :REM Broadcast operation
530  $(tx_blk%+4)="BRIDGE"
540  tx_blk%?10=reply_port
550  tx_blk%?11=network%
560  X%=tx_blk%:Y%=tx_blk% DIV 256
570  A%=&10:CALL osword
580  A%=&32
590   REM Wait for completion
600  X%=FNpoll_tx
610  redo=redo-1
620 UNTIL redo=0 OR X%=&40 OR X%=&43 OR X%=&44 OR X%=0
630 IF X%>0 THEN PRINT"Broadcast failed, error &";~X%:PROCdelete:END
640ENDPROC
650
660DEF PROCdelete
670 REM delete a receive block
680 A%=&34
690 X%=blk%?0
700 CALL osbyte
710ENDPROC
720
730DEF FNpoll_tx
740 REPEAT UNTIL NOT ((USRosbyte) AND &8000)
750=(((USRosbyte) AND &FF00) DIV &100)
760
770DEF PROCdisplay(no%)
780 PRINT"Found network ";no%
790 PROCdelete
800 PROCset_up_rx
810ENDPROC
```

10.19 Printers

The protocol used by SJ Research to locate named printer resources is an adaptation of the protocol used by Acorn (and implemented in ANFS). The extra complexity is introduced by the possibility of having multiple printers at the same station, such that the printer server has to keep track of the current printer selection for each user. The printer status protocol is used : a status request is broadcast, containing the name of the printer desired, and the station, making the broadcast can chose which of the responding servers to select.

Data for printing is sent to the print server in packets, using port &D1. Each packet has a maximum length of 80 bytes. All packets received by the print server (including logon/ logoff packets) are acknowleged by sending a packet back to the client, when the server is ready to receive the next data packet. These ack packets are sent on port &D1, and are 1 byte long (this byte contains no information).

All packets have a sequence number in bit 0 of the control code. The purpose of this is to discard duplicate packets, which occur due to a fundamental problem with networks. It is possible for a packet to be sent and received correctly, but the transmitter failed (and so tries again) while the receiver is unaware of the problem and receives both copies as good packets. The solution is to attach a sequence number to each packet, and when two packets are received with the same sequence number, the second can be discarded. It turns out that a single bit is the control code : when repeating packets after a failed transmit, the sequence numbers should be kept the same. Ack packets are sent with the same sequence number as the data packet being acknowleged : when waiting for an ack packet, any that arrive with a different sequence number from the last data packet should be ignored.

A client logs on to the print server by sending a data packet to the print server with the control code = &82. This packet should contain one byte, value zero (although there is a bug in Acorn NFS whereby a number of ctrl-C characters may be sent in front of the zero). Further packets follow, containing the data to be printed, with a control code = &80+sequence. The client logs off by sending a packet with control code = &84+sequence bit. This packet may contain print data, but the last byte is discarded (it is assumed to be the ctrl-C from a BBC micro).

Problems with printing

With NFS 3.34 printing anything other than straight text is not guaranteed. This is because it is not possible to send the characters 2 and 3 to the printer without them being interpreted specially, even by using VDU 1. There is another problem with the NFS inserting extra 'null' characters into the print stream.

With NFS 3.6 a new print protocol has been devised which allows any character to be sent to the Print Server, thus allowing graphics dumps to be printed.

When using the Econet Print Server it is necessary to understand the concept of logging-on and -off (The same as with an FS). Basically VDU 2 logs on, and VDU 3 logs off: on non-spooling Print Servers (eg the Acorn PS ROM, some SJ File Servers) logging-on has the effect of 'locking-out' other users so that they get a 'Not listening' error. Logging-on also has the effect of printing a banner, including possibly time of printing and the user's name, whilst log on & off flushes the print buffer and sends a form-feed. Therefore these two commands should **not** be used to enable/disable output to a printer (when some characters must go to to screen but not the printer), even though this works OK with a local printer.

To enable/disable printer output after a VDU 2, use *FX3 4 to disable the printer and *FX 3 0 to enable it again. See page 442 of the BBC User Guide.

10.20 File Server Interface

Introduction

This section deals with the interfaces through which clients (station users and user programs) can interact with the File Server. The interfaces allow a client to manipulate the filestore, to gather information about files, directories, other users of the file server and about the configuration of the file server.

The BBC microcomputer uses the fileserver interface for all filing system operations, including '*' commands, LOAD and SAVE. When a command is sent from the BBC microcomputer to the file server, it will first be converted to transmit packets. For example if the line ***I AM 0.235 TONY** was typed, this would first set my fileserver number to 235 on my local network. It would then convert the string to a fileserver packet with a function code of 0 (OSCLI command) and send the packet using a control byte of &80 on port &90. The file server expects all commands to be send to it on port &90, the network filing system ROM in the BBC will specify port &99 for the file servers to reply.

Once the BBC microcomputer has finished sending the data, the fileserver will send a reply packet. For a logon packet this will contain three context handles and a boot option. The NFS ROM in the BBC will convert these to its own internal format.

Conventions and Abbreviations

All numbers used are given in decimal except where prefixed by '&' to indicate hexadecimal notation. CR is used as an abbreviation for 'carriage return', character &0D.

The User Environment

A client is identified and authenticated to the file server by its station number and three HANDLES. These handles are created by the file server by opening directories, and identify to the file server the environment in which to interpret commands and to look-up filenames presented by the client. The handles are created by the file server when a user logs on and are closed when the user logs off again. The three handles which comprise the user environment are:

The Currently Selected Directory (CSD) - the directory in which to look up all filenames which do not refer explicitly to another directory.

The User Root Directory (URD) - the directory where a user is placed after logging-on and where he is returned after a *DIR command. On SJ Research fileservers, the character '&' can be used to specify the URD within commands.

The Library (LIB) - the directory where unrecognised commands are looked up if the filename is not found in the CSD.

Usually the client machine software deals with the manipulation of these handles; but it is possible for a user to define his own environment by opening several directories, and declaring a set of these handles as representing the current environment. This enables the user to execute a command in a number of different environments.

Almost all file server commands need these three handles to be transmitted as part of the information.

File Server Function Codes

These calls will be used for all communication with the file server. If these calls are made directly to the file server, in particular *DIR and *I AM, then it is possible that the software will not work with all versions of the NFS ROM, in particular the ANFS, because the filing system ROM will not be able to update its workspace. The control block outlined below is always sent to your currently selected fileserver; to change your fileserver number use the OSWORD call with A=&13, reason code=1.

Listed under each File Server call is the packet to be sent to the fileserver. The most convenient way to send packets to the fileserver is by use of OSWORD A=&14, which automatically inserts context handles and controls the packet transmission/reception. To use this call, a standard header must be added to the other information, as shown below.

On Entry,

A=&14
The NFS must be the current filing system.
YX point to the address of the transmit control block, which is shown below :-

0	0
1	Size of whole block (n)
2	0
3	Function code
4	0
7	Rest of File server transmit block
n	

When the File Server call is made, the NFS ROM modifies the control block, as shown below, before sending it to the file server. This allows the user to communicate with the File Server without having to know his context handles, or which port he should communicate with the File Server on.

Bytes 0 to 7 are referenced throughout this section as the *transmit block*. Bytes 2 to n are the bytes that are actually sent through the network, to the File Server.

0	0
1	Size of whole block (n)
2	Reply port
3	Function code
4	User's Root Directory context handle in File Server format
5	Currently Selected Directory context handle in File Server format
6	Currently Selected Library context handle in File Server format
7	Rest of File server transmit block
n	

On Exit,

A,X,Y undefined

The transmit control block is replaced by the receive control block shown below :-

0	0
1	Size of whole block (new value)
2	Command Code
3	Return Code
4	Rest of File server Receive block
x	

Bytes 0 to 4 are referenced throughout this section as the receive block. Bytes 2 to 4 are the bytes that are actually returned by the file server.

The *command code* indicates to the client what action (if any) the client should take upon receiving this response.

The *return code* is an indication to the client of any error status which has arisen, as a result of attempting to execute the command. A return code of zero indicates that the command step completed successfully; otherwise the return code is the error number indicating what error has occurred. If the return code is non-zero, then the remainder of the message contains an ASCII string terminated by a carriage return, which describes the error.

Summary of File Server Calls

Function code Description

0	Command line decoding
1	Save
2	Load
3	Examine
4	Catalogue header (Acorn only)
5	Load as command
6	Open file
7	Close file
8	Get byte
9	Put byte
10	Get bytes
11	Put bytes
12	Read random access information
13	Set random access information
14	Read disc name information
15	Read logged on users
16	Read date/time
17	Read EOF (end of file) information
18	Read object information
19	Set object information
20	Delete object
21	Read user environment
22	Set user's boot option
23	Logoff
24	Read user information
25	Read file server version number
26	Read file server free space
27	Create directory, specifying size
28	Set time/date
29	Create file of specified size
30	Read user free Space (Acorn only)
31	Set user free Space (Acorn only)
32	Read client user identifier
64	Read account information (SJ Research only)
65	Read/write system information (SJ Research only)

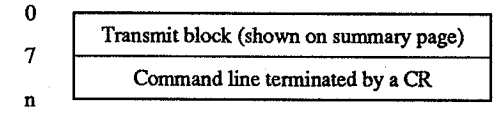
Command Line Decoding

Function code=0

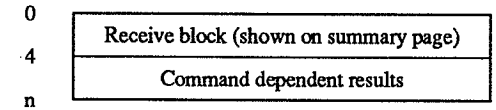
General description

A number of the operations performed by the file server are initiated by the sending of a command line. The function code of zero indicates to the file server that such a command line has been received. All command line type exchanges have the same format:

Client (command port):



File Server (reply port):



If the command requires more action by the client, then the command code indicates what command line the file server has decoded. The file server will also return any decoded parameters or data which the client will need to complete the command. The possible command codes that the file server may return are:-

0	No Action, command complete
1	*Save
2	*Load
3	*Cat
4	*Info, *Printer, *Printout
5	*I AM
6	*Sdisc (Acorn only)
7	*Dir, *Sdisc (SJ Research only)
8	Unrecognised command
9	*Lib

Description of returned command codes

The following receive blocks are the result of command line decoding done by the file server. Any context handles returned by the file server are in file server internal format.

*SAVE Command code 1

0	Receive block (shown on summary page)
4	File load address
8	File execute address
12	File size
15	File name terminated by a CR
n	

The protocol then continues with function code 1

*LOAD Command code 2

0	Receive block (shown on summary page)
4	File load address
8	Flag : If flag=&FF then load address is defined
9	File name terminated by a CR
n	

The protocol then continues with function code 2.

*CAT Command code 3

0	Receive block (shown on summary page)
4	Decoded directory name terminated by CR
n	

*INFO Command code 4

0	Receive block (shown on summary page)
4	Character string of information terminated by negative byte (&80)
n	

*I AM Command code 5

0	Receive block (shown on summary page)
4	URD handle
5	CSD handle
6	LIB handle
7	Boot option (4 least significant bits)
8	

*SDISC Command code 6

0	Receive block (shown on summary page)
4	URD handle
5	CSD handle
6	LIB handle
7	

*DIR Command code 7

0	Receive block (shown on summary page)
4	CSD handle
5	

Unrecognised command Command code 8

0	Receive block (shown on summary page)
4	Command string terminated by CR
n	

*LIB Command code 9

0	Receive block (shown on summary page)
4	LIB handle
5	

Note that CSD, URD and LIB handles returned by the File Server are not suitable for selecting with OSWORD A=&13 as the NFS maps these onto a set of internal handles. Hence these commands should usually be issued via the operating system.

SAVE

Function code=1

General description

This call requests to save a file. This call will be made following a *SAVE command line sent to the file server (used by the Acorn SYSTEMs and Atoms). Note that the BBC computer interprets the paramters to a *SAVE command internally and will enter the protocol by issuing a save with function code 1.

Client (command port):

0	Standard Transmit block except data acknowledge port replaces URD handle
7	file load address
11	file execute address
15	file size
18	file name terminated by CR
n	

File server (reply port):

0	Standard Rx Header
4	data port
5	maximum data block size
7	file name terminated by CR (only present if command code is 3)
n	

The client and file server now enter the 'data exchange phase' of the protocol, where the file server acknowledges the receipt of each data packet. If there is no data to be sent (eg a zero length file) then this phase is omitted. If the file server detects an error during the data transfer phase (eg a disc error), then the 'data exchange phase' is allowed to complete but the SAVE operation is aborted. The error status is returned in the return code of the 'final acknowledge'. Because the data is not transferred on the command port the following data must be sent by direct transmissions to the file server.

Client (data port):

0	A block of data, up to the maximum data block size.
n	

File server (data acknowledge port):

0	Undefined
1	

When the final data block has been received by the file server, the 'data acknowledge' is replaced by the 'final acknowledge', which is the terminating packet of the protocol.

File server (reply port):

0	Command code
1	Return code
2	Access byte (in standard format)
3	file creation date
5	

LOAD

Function code=2

General description

This call requests to load a file. This call will be made following a *LOAD command line sent to the file server (used by the Acorn SYSTEMs and Atoms). Note that the BBC computer interprets the paramters to a *LOAD command internally and will enter the protocol by issuing a load with function code 2. This call will not work via OSWORD A=&14.

Client (command port):

0	Standard Transmit block with byte 4 containing dataport
7	File name terminated by CR
n	

File server (reply port):

0	Standard Rx Header
4	file load address
8	file execute address
12	file size
15	file access (as for SAVE)
16	file creation date (as for SAVE)
18	file name terminated by CR
n	

If the file is of zero length then the 'data transfer' phase is omitted. If the file server detects an error (eg disc error), then the required amount of data will be sent but its data content is undefined. Because the data is not transferred on the command port the following data must be sent, and received by direct transmission and reception.

Data is transmitted until 'file size' data has been sent.

File server (data port):

0 n	Data block

File server (reply port):

0	Command code
1	Return code
2	

Examine

Function code=3

General description

This call returns information about files in a particular directory. The detail of the information is dependent on the value of ARG.

0 - All information, machine readable format
1 - All information, character string
2 - File title only
3 - File title and access, character string

Client (command port):

0	Transmit block (shown on summary page)
7	ARG
8	Entry pointer to directory
9	Number of objects to examine
10	Directory name (terminated by a <CR>)
n	

The directory entry pointer gives the entry number within the directory from which to examine. Conventionally the first entry in a directory is entry number zero.

The number of entries to examine specifies how many entries are to be examined, so is usually determined by the buffer space available to the client. A parameter of zero in this case conventionally demands that all entries in the directory from the entry point to the end of the directory be examined. On an SJ Research File Server if 0 entries are requested then no entries are returned.

Where information is returned in character string format, individual entries are delimited by zero bytes (&00), the final entry being terminated by a negative byte (&80). Carriage returns may occur within such strings. Where data is returned in machine readable format, the entries consist of a defined number of bytes; and so there are no delimiters between entries although the final entry is terminated by a negative byte.

On exit,

If ARG=0

0	Receive block (shown on summary page)
4	No. of entries examined
5	Cycle number
6	Object's title (Padded with spaces)
16	Reload address
20	Execution address
24	Access (bits 7-0: M P D L W R w r)
25	Date object last updated
26	Year & month object last updated (top 4 bits = year since 1981, bottom 4 bits = month)
27	Main account number (low byte)
28	Top 3 bits are main account number Bottom 3 bits are aux. account number
29	Aux. account number (lo byte)
30	Size of file (in bytes)
33	Next file (as above, bytes 5-31)
n	

Note that the Acorn file server will return its system name in bytes 27 to 30. If an entry in the print queue has been examined then the access D/R is a /spl file and an access D/WR is a /prt file.

If ARG=1,

0	Receive block (shown on summary page)
4	No. of entries examined
5	Cycle number
6	Chararacter string of all information
n	

If ARG=2

0	Receive block (shown on summary page)
4	10 (object name length for BBC MOS)
5	object name padded with spaces
n	

If ARG=3

0	Receive block (shown on summary page)
4	object name padded to 10 characters followed by formatted access string
22	

Data terminated by an &80.

Catalogue Header

Function code=4

General description

This call returns the data used in the *CAT command.

On entry,

0	Transmit block (shown on summary page)
7	Directory name (CR returns information for CSD)
n	

On exit,

0	Receive block (shown on summary page)
4	Last component of directory name padded with spaces
14	Character indicating ownership of directory
15	three space characters (&20)
18	Current disc name padded with spaces (Terminated by CR, negative byte
33	

Note that this call is only fully supported by Acorn File Servers as it is only used by Systems 3/4/5 and Atoms.

Load as Command

Function Code=5

General description

On entry,

0	Transmit block (shown on summary page)
7	file name (Terminated by CR)
n	

The remainder of this protocol is exactly as for function code 2 (LOAD), except the file name is looked up first in the CSD, and if not found then, also in LIB. It is used for loaded '*' commands. The error returned if the file name is not found in either directory is 'Bad command'. On an SJ Research File Server the call is equivalent to function code=5 except for run only users.

Find (OPEN)

Function Code=6

General description

This function code creates a handle for the object specified, with the access type requested. Such handles are used for performing random access operations and also for manipulating the user's environment. An object will be opened only if the client has the necessary access rights to the object. A file can be opened several times for reading only, but only once for update. A file will be created if:

1: The file does not exist
2: The file is opened for update

On entry,

0	Transmit block (shown on summary page)
7	0 = create a new file (delete data in an existing file) non-zero = object must already exist
8	0 = open object for update non-zero = open object for read only
9	object name terminated by CR
n	

On exit,

0	Receive block (shown on summary page)
4	file handle
5	

Close

Function Code=7

General description

This function indicates to the file server that the handle passed as argument is no longer needed, and that all of the updated data in the file should be written out to the disc. A handle of zero indicates to the file server that all handles to open FILES are to be closed. This call does not close handles to directories. Note that the handle is a File Server format handle.

On entry,

0	Receive block (shown on summary page)
4	Handle
5	

On exit,

0	Receive block (shown on summary page)
4	

BGET

Function Code=8

General description

The next four function codes deal with the facilities that the file server provides to enable the user to perform random access operations on open files. These operations have an additional protocol to ensure the integrity of the data exchanged, provided by a SEQUENCE NUMBER. The sequence number is a single bit, held in both client station and file server, which differentiates between successive reads of a file using the pointer held in the file server, and repeated reads of the same byte because the operation failed at the first attempt. The sequence number is sent in the least significant bit of the flag byte of the Econet control block. The file server will return its copy of the sequence number with the data to allow the client to detect data sequencing errors.

The client should invert his copy of the sequence number after every successful transaction with the file server. If the client detects a data packet with the incorrect sequence number then the client should be prepared to repeat the request.

This function code reads a byte from the file at the position specified by the file server's internal file pointer. Note that the handle is a File Server format handle.

On entry,

0	0
1	Size of whole block (5)
2	Reply port
3	Function code (8)
4	File handle
5	

On exit,

0	Receive block (shown on summary page)
4	byte read &FE if reading first byte after file end
5	flag byte &00 = normal read operation &80 = last byte in the file &C0 = first byte after file end
6	

BPUT

Function Code=9

General description

This function code writes a single byte to the file at the position indicated by the file server's internal file pointer. Note that the handle is a File Server format handle.

On entry,

0	0
1	Size of control block (6)
2	Reply port
3	Function code (9)
4	File handle
5	Byte to be written
6	

On exit,

0	Receive block (shown on summary page)
4	

Getbytes and Putbytes

Function Code=10 and 11

General description

These operations allow the client to access blocks of data. The client may supply an offset within the file at which to start the operation, or may use the sequential file pointer maintained by the file server. The protocol includes a sequence number as described for BGET and BPUT.

On entry,

0	0
1	Size of rest of block
2	0
3	Function Code
4	Data Ack Port
5	CSD handle
6	LIB handle
7	File Handle
8	non-zero = use FS sequential pointer Zero =use supplied offset
9	Number of bytes
12	File offset (if supplied)
15	

Reply to the first call is :-

0	Receive block (shown on summary page)
4	Data port (function code 11 only)
5	Maximum data block size (function code 11 only)
7	

The data transfer phase is exactly as described for LOAD and SAVE. For transfers of zero data these steps are not executed. If a read extends over the end of the file then the requested amount of data will be returned, but the number of valid data bytes will be less than the amount of data requested. The remaining data is undefined.

On exit,

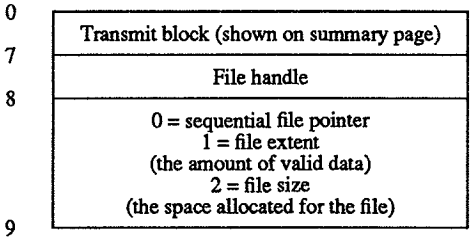
0	Receive block (shown on summary page)
4	0 = all ok &80 = read includes last byte in file (Value undefined for PUTBYTES)
5	Number of valid data bytes transferred
8	

Read Random Access Information Function Code=12

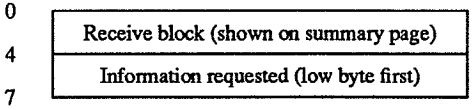
General description

This function code allows the client to discover information about files for which he currently has handles.

On entry,

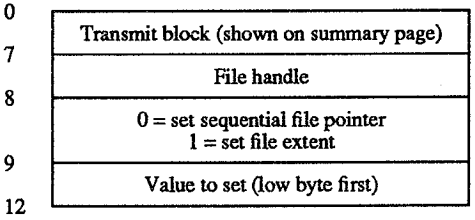


On exit,

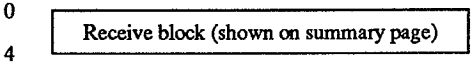


Set Random Access Information Function Code=13

On entry,



On exit,



Setting the extent of a file is not supported on some early File Servers.

Read Disc Information

Function Code=14

General description

This function returns the disc configuration of the file server. Conventionally the first drive in the file server has drive number zero. If the number of drives to interrogate is zero then the file server will return information on all drives in the system. It is not necessary to be logged on to read this information.

On entry,

0	Transmit block (shown on summary page)
7	
7	First drive number
8	
8	Number of drives to interrogate
9	

On exit,

0	Receive block (shown on summary page)
4	
4	Number of drives found
5	
5	Drive number of first drive selected
6	
6	Disc name padded with spaces
22	
22	Drive number of second drive selected
23	
23	.
	.
	.
n	

Read Current Users

Function Code=15

General description

This function returns the currently logged-on users of the file server, their station numbers and associated privileges. Conventionally the logged-on user entries start at zero. A request for zero entries will return information for all users, commencing at the start entry.

On entry,

0	Transmit block (shown on summary page)
7	
7	Start entry
8	
8	Number of entries to get
9	

On exit,

0	Receive block (shown on summary page)
4	
4	Number of entries returned
5	
5	Machine number of first user (low byte, high byte)
7	
7	First user name terminated by CR
n	
n	0 = User not privileged non-zero = user privileged
(n+1)	
(n+1)	Second user name terminated by CR
	.
	.
	.

Read Date and Time

Function Code=16

General description

It is not necessary to be logged-on to the file server to use this function code.

On entry,

0	Transmit block (shown on summary page)
7	

On exit,

0	Receive block (shown on summary page)
4	Date 1st byte days 2nd byte (top 4 bits) - years since 1981 (low 4 bits) - month
6	Time 1st byte hours 2nd byte minutes 3rd byte seconds
9	

If the call is performed on an Acorn level 2 File Server without a time board then the three bytes containing the time will be set to 0. Beware that this software can misinterprated this result by reading the time at midnight and deducing that no time board is attached!

Read 'End-of-file' status

Function code=17

General description

This function is valid for file handles only.

On entry,

0	Transmit block (shown on summary page)
7	File handle
8	

On exit,

0	Receive block (shown on summary page)
4	0 = file pointer within file &FF = file pointer outside file
5	

Read Object Information

Function Code=18

General description

This call returns detailed information about a file or directory. The File Server call with ARG=64 is only supported on the SJ Research file server.

On entry,

Client (command port):

0	Transmit block (shown on summary page)
7	ARG 1 = Object creation date 2 = Load and execute address 3 = Object extent (three bytes only) 4 = Access byte (as for EXAMINE) 5 = All object attributes 6 = Access and cycle number of dir. 64 = Read creation and update date.
8	Object name terminated by CR
n	

On exit,

The results returned depend on the argument passed with the call.

ARGs 1-5

File server (reply port):

0	Receive block (shown on summary page)
4	0 = object not found 1 = object is a file 2 = object is a directory
5	Results returned (load/exec/size/access/date/ownership) ownership = &00 -> owner = &FF -> public
n	

ARG 6

File server (reply port):

0	Receive block (shown on summary page)
4	Undefined
5	0
7	Directory name padded with spaces
17	Access to directory &00 = owner access &FF = public access
18	Cycle number of directory
19	

ARG 64

File server (reply port):

0	Receive block (shown on summary page)
4	0 = object not found 1 = file found 2 = directory found
5	Creation date in standard format Byte 5 - Day Byte 6 (Top 4 bits) - Year since 1981 (Bottom 4 bits) - Month Byte 7 - Hour Byte 8 - Minutes Byte 9 - Seconds
10	Last update in standard format Byte 10 - Day Byte 11 (Top 4 bits) - Month (Bottom 4 bits) - Year since 1981 Byte 12 - Hour Byte 13 - Minutes Byte 14 - Seconds
15	

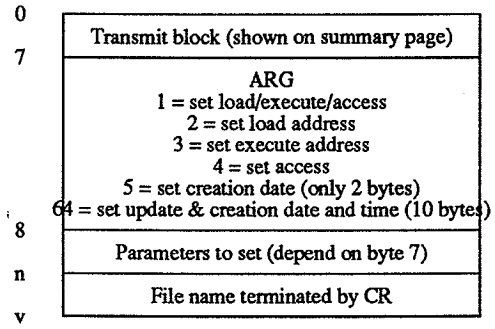
Set Object Attributes

Function Code=19

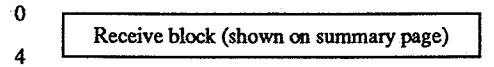
General description

This call is used to set the attributes for a file or directory. This is used by OSFILE. Note ARG=64 is only supported on SJ Research File Servers.

On entry,



On exit,



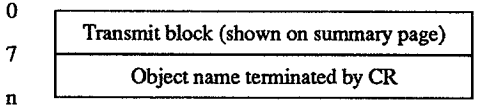
Delete Object

Function Code=20

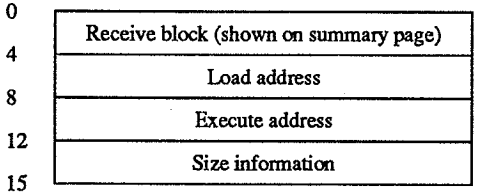
General description

This call is used to delete an object and is used by OSFILE. It is not possible to use wildcards with this call.

On entry,



On exit,



Read User Environment

Function Code=21

General description

This call returns your current position on the file server.

On entry,

0	Transmit block (shown on summary page)
7	

On exit,

A,X,Y

0	Receive block (shown on summary page)
4	Length of disc name (16)
5	Name of currently selected disc padded with spaces
21	Name of CSD padded with spaces
31	Name of LIB padded with spaces
41	

Set User Boot Option

Function Code=22

General description

This call writes your boot option to the password file. This call is used by *OPT 4.

On entry,

0	Transmit block (shown on summary page)
7	boot option
8	

On exit,

0	Receive block (shown on summary page)
4	

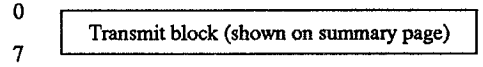
User log-off

Function Code=23

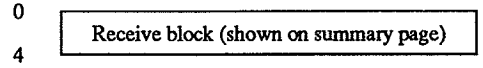
General description

This call is equivalent to a *BYE packet to the fileserver, except that it closes open print files as well. The File Server, on receiving the command, removes the station number from its internal table, closes all file handles and context handles open for that station number.

On entry,



On exit,



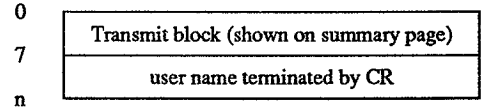
Read User Information

Function Code=24

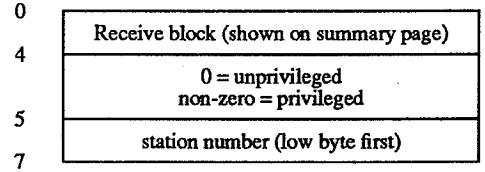
General description

This call returns the first station number in its user table that matches the user identifier provided. Note that on an SJ Research file server the station number returned will always be the machine which the user most recently used, this is not the case with the Acorn file server.

On entry,



On exit,



Read FS Version Number

Function Code=25

General description

This call returns a version number string. It is not necessary to be logged onto the file server to do this call. This function is used in the SJ Research library utility FSLIST.

On entry,

0	Transmit block (shown on summary page)
7	

On exit,

0	Receive block (shown on summary page)
4	Character string describing version number
n	

Read FS Free Space

Function Code=26

General description

This call returns the size and current amount of free space on a file server disc. This call is used by the library utility *FREE.

On entry,

0	Transmit block (shown on summary page)
7	Disc name (terminated by a CR)
n	

On exit,

0	Receive block (shown on summary page)
4	Free space on disc (low byte first)
7	disc size (low byte first)
10	

Create Directory

Function code=27

General description

This call creates a directory specifying the number of blocks, for the file server, to allocate to that directory. One block is 256 bytes.

On entry,

0	Transmit block (shown on summary page)
7	Number of blocks to allocate
8	Directory name (Terminated by a <CR>)
n	

On exit,

0	Receive block (shown on summary page)
4	

Set Real time clock

Function code=28

General description

This System privileged command allows the Date and Time to be set, and is used in the Library utility program 'settime'. This command is normally privileged on SJ Research File Servers, but this may be changed, by using function code 65.

On entry,

0	Transmit block (shown on summary page)
7	Date in standard format Byte 7 - Day Byte 8 (Top 4 bits) - Year since 1981 (Bottom 4 bits) - Month
9	Time in standard format Byte 9 - Hour Byte 10 - Minutes Byte 11 - Seconds
12	

On exit,

0	Receive block (shown on summary page)
4	

Create file

Function code=29

General description

This follows the same protocol as Command code 1, 'SAVE'. However the data transfer phase is omitted. The result is that a requested amount of space is reserved for a file, the data therein being undefined. This call is not supported on early versions of File Server software.

Read User Free Space

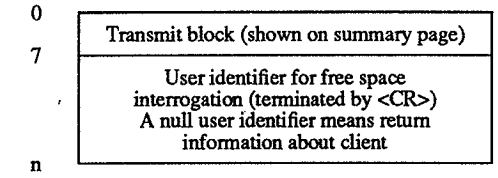
Function code=30

General description

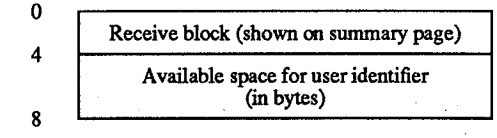
This call is the Acorn file server call to read a users free space, the call is the equivalent of the SJ Research accounting system (detailed in function code 64).

On entry,

Client (command port),



On exit,



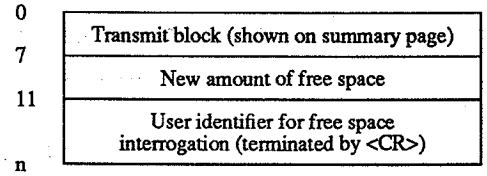
Set User free space

Function code=31

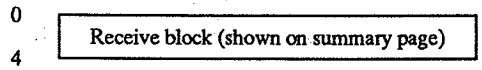
General description

The function sets the amount of space available for a user identifier. The function is only legal for system privileged users. The user identifier is that of the client whose space allocation is to be ammended. This call is only implemented on Acorn File Servers.

On entry,



On exit,



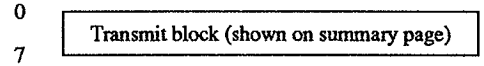
Read client User Id.

Function code=32

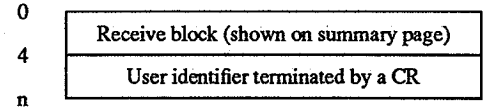
General description

Reads the username which you used to logon to the File Server with. This call is not supported on early version of File Server software.

On entry,



On exit,



Read Account information

Function code=64

General description

This call reads the amount of space left in an account.

On entry,

0	Transmit block (shown on summary page)
7	ARG=0
8	First account to try
10	Maximum number of accounts to send information on
12	Disc number
13	

On exit,

0	Receive block (shown on summary page)
4	Next account to try
6	Number of accounts returned
8	1st account number
10	1st account space
12	2nd account number
14	2nd account space
16	. . .

N.B. This call is only supported on the SJ Research file server.

Read/write system info.

Function code=65

General description

This call provides an interface to read and write the printer information and change the privilege needed to write the file server's time. All of these read calls are unprivileged commands. All the write operations are privileged.

Reset print server information :

This call is used to reset the printer information and **must** be issued for any of the other change printer information calls to take effect.

On entry,

0	Transmit block (shown on summary page)
7	ARG=0
8	

On exit,

0	Receive block (shown on summary page)
4	

Read current state of printer :

This call returns the detailed information about a logical printer.

0	Transmit block (shown on summary page)
7	ARG=1
8	Printer number (1 - 8)
9	

Read/Write System Info.

Function code=65

General description

This call provides an interface to read and write the printer information and change the privilege needed to write the file server's time. All of these read calls are unprivileged commands. All the write operations are privileged.

Reset print server information :

This call is used to reset the printer information and **must** be issued for any of the other change printer information calls to take effect.

On entry,

0	Transmit block (shown on summary page)
7	
8	ARG=0

On exit,

0	Receive block (shown on summary page)
4	

Read current state of printer :

This call returns the detailed information about a logical printer.

0	Transmit block (shown on summary page)
7	
8	ARG=1
9	Printer number (1 - 8)

On exit,

0	Receive block (shown on summary page)
4	
10	Name of printer (padded with spaces)
	Bit 3 - Spool to disc Bit 2 - Account ownership required Bit 1 - Anonymous users allowed Bit 0 - Printing enabled
11	Account number (Only relevant if bit 2 of previous byte is set)
13	Banner file name (terminated by a CR if less than 23 characters)
n	

Write current state of printer

This call writes the detailed information about a printer, system privilege is required to do this. This call is not supported on SJ Fileservers version 1.00 or greater; see ARG=16 for call to write printer information.

On entry,

0	Transmit block (shown on summary page)
7	
8	ARG=2
9	Printer number (1 - 8)
15	Name of printer (padded with spaces)
	Bit 3 - Disable spooling to disc Bit 2 - Account ownership required Bit 1 - Anonymous users allowed Bit 0 - Printing enabled
16	Account number (Only relevant if bit 2 of previous byte is set)
18	Banner file name (terminated by a CR if less than 23 characters)
n	

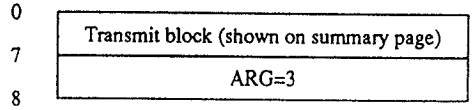
On exit,

0	Receive block (shown on summary page)
4	

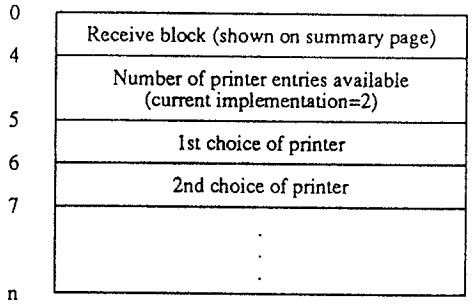
Read the AUTO printer priority

This call reads the order in which printers are selected for users who have not requested a particular printer. This call is not supported on SJ Fileservers version 1.00 or greater.

On entry,



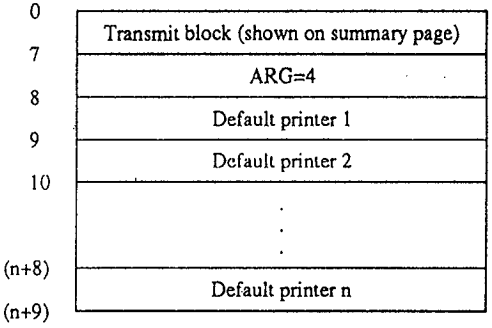
On exit,



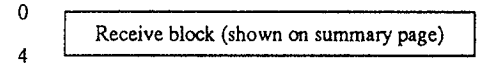
Write the AUTO printer priority

This call allows a privileged user to write the order in which printer are selected for users who have requested the AUTO printer. This call is not supported on SJ Fileservers version 1.00 or greater.

On entry,



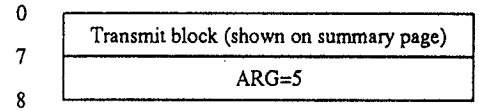
On exit,



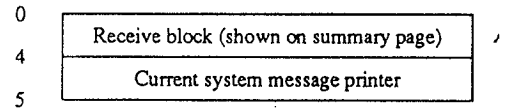
Read system message channel

This call returns the physical printer that all system messages are sent to. Note that the printer is a physical printer, so the parameter should be either 1 (parallel) or 2 (serial).

On entry,



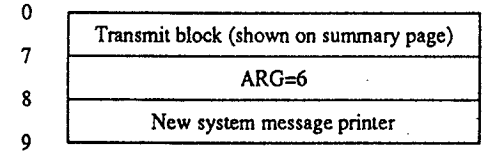
On exit,



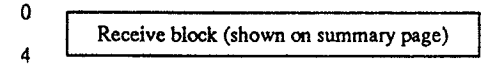
Write system message channel

This call allows a privileged user to set the physical printer that system messages come out of.

On entry,



On exit,

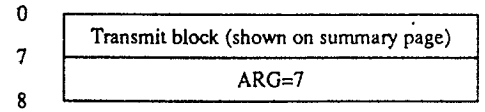


Read message level

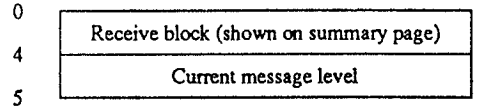
This call reads the current level of system messages. The value returned is in the range of 0 to 255. The amount of output is the level of output selected plus all the levels below that level. Therefore, in the list of levels shown to set the message level to 7 would make the file server print all logons and logoffs as well as errors.

Message level	Description
0	Off
5	Logon/logoff
7	Errors (i.e. 'Wrong password', 'bad name' etc.)
10	Maximum users and all star commands
11	Load/save
15	*cat and opens
128	Aborted loads
130	Function codes
150	Network errors
170	Map building names
200	Disc read/write
250	All successful network transactions to and from the fileserver
255	All Activity to the JPROC processor

On entry,



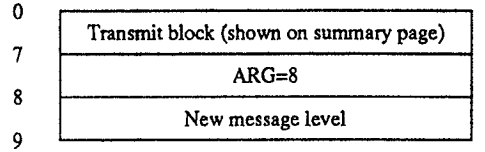
On exit,



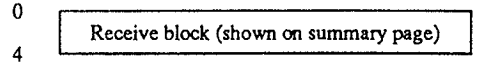
Set message level

This call sets the message level, as described above. It should never be neccessary to set the message level to greater than 127 and that setting the message level to a value greater than 150 produces excessive output and will probably reduce the performance of the file server.

On entry,



On exit,



Read the default printer

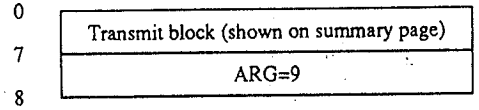
This call returns the default printer. This printer will be selected if a user starts to print without having selected a particular printer. This call is not supported on SJ Fileservers version 1.00 or greater.

The values for the default printer are :-

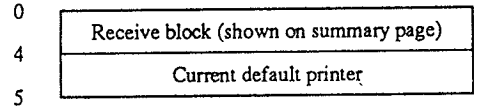
0 Automatic search through the list default printer priority list (set by Fn=65 ARG=4)
1 Logical printer 1
2 Logical printer 2
. .
. .
. .
8 Logical printer 8

255 Hold the job output in the %PRINTQ directory.

On entry,



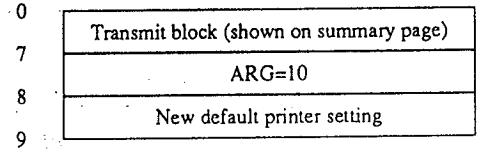
On exit,



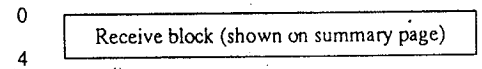
Set the default printer

This privileged call sets the default printer, see ARG=9 for more information about valid printer numbers.

On entry,



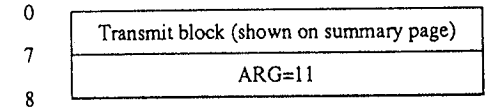
On exit,



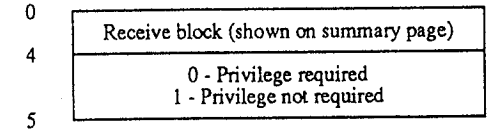
Read the privilege required to change time

This call reads the privilege required to change the file servers time. The SJ Research file server normally insists on system privilege to be able to change the time. However, if it is desired to change the time frequently this can be disabled.

On entry,



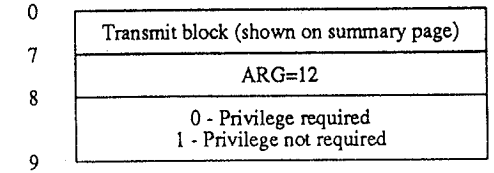
On exit,



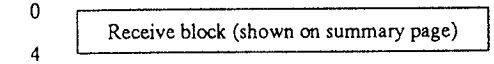
Set the privilege required to change time

This call sets the privilege required to change the file servers time.

On entry,



On exit,



Read printer information

This call is only available on SJ File Servers version 1.00 or greater. This one call replaces the several calls previously needed to read all printer information.

On entry,

0	Transmit block (shown on summary page)
7	ARG=15
8	Number of printers on which to return information
9	Start printer number (0-15)
10	

On exit,

0	Receive block (shown on summary page)
4	Number of printer entries returned
5	Printer name
11	Status of printer 0-Off 1-Spooling 2-Non-spooling 3-Hold 4-Auto
12	1-Default printer 0-Not a default printer
13	1-Anonymous users allowed 0-No anonymous users allowed
14	1-Account number required 0-No account number required
15	Account number (Only relevant if previous byte was 1)
17	Output port 1-Parallel 2-Serial or 1st choice printer no. for Auto
18	2nd choice printer no. for Auto
19	Reserved
21	Next printer (as above bytes, 5-20)
n	

Write printer information

This call is only available on SJ File Servers version 1.00 or greater. This one call replaces the several calls previously needed write all printer information.

On entry,

0	Transmit block (shown on summary page)
7	ARG=16
8	Number of printers for which to write information
9	Start printer number (0-15)
10	Printer name
16	Status of printer 0-Off 1-Spooling 2-Non-spooling 3-Hold 4-Auto
17	1-Default printer 0-Not a default printer
18	1-Anonymous users allowed 0-No anonymous users allowed
19	1-Account number required 0-No account number required
20	Account number (Only relevant if previous byte was 1)
22	Output port 1-Parallel 2-Serial or 1st choice printer no. for Auto
23	2nd choice printer no. for Auto
24	Reserved
26	Next printer (as above bytes, 10-25)
n	

On exit,

0	Receive block (shown on summary page)
4	

Read Encryption Key

Function code=66

General description

This call supplies an encryption key.

Read/Write Backup

Function code=67

General description

This call provides an interface to read and write information necessary for an automatic backup. The call can also be used to read the tape id block. If there is no error the backup is possible.

Determine whether backup is possible

This call is to determine whether a tape backup is currently possible.

On entry,

0	Transmit block (shown on summary page)
7	ARG=0
8	

On exit,

0	Receive block (shown on summary page)
4	

Read tape id block

This call reads data from the tape Id block of the tape currently loaded.

On entry,

0	Transmit block (shown on summary page)
7	ARG=1
8	offset from start of block
10	number of bytes to return (should be <128)
12	

On exit,

0	Receive block (shown on summary page)
4	Data from tape Id block
n	

Read current status of auto backup

This call returns the details of any auto backup currently pending.

On entry,

0	Transmit block (shown on summary page)
7	ARG=2
8	

On exit,

0	Receive block (shown on summary page)
4	0-No backup pending 1-Backup pending
5	Time of backup in standard format Byte 5 - Day Byte 6 (Top 4 bits) - Years since 1981 Byte 6 (Bottom 4 bits) - Month Byte 7 - Hour Byte 8 - Minutes Byte 9 - Seconds
10	Printer output 0 - Off 1 - Parallel 2 - Serial
11	Tape partition for backup currently set to 0
12	Disc name to backup (terminated by a CR if less than 10 characters)
22	Tape name to use for backup (terminated by a CR if less than 10 characters) A null name means any tape can be used
32	

Write current status of auto backup

This call is used to set an auto backup or to cancel a previously defined auto backup.

On entry,

0	Transmit block (shown on summary page)
7	ARG=3
8	0-Cancel backup 1-Set a new backup (Following bytes are only relevant if setting a backup)
9	Time of backup in standard format Byte 9 - Day Byte 10 (Top 4 bits) - Years since 1981 Byte 10 (Bottom 4 bits) - Month Byte 11 - Hour Byte 12 - Minutes Byte 13 - Seconds
14	Printer output 0 - Off 1 - Parallel 2 - Serial
15	Tape partition for backup currently set to 0
16	Disc name to backup (terminated by a CR if less than 10 characters)
26	Tape name to use for backup (terminated by a CR if less than 10 characters) A null name means any tape can be used
36	

On exit,

0	Receive block (shown on summary page)
4	

N.B. These calls are only supported on the SJ Research file server.

10.21 Tape Id Block Format

0	"SJ Research"
11	0=blank 1=used
12	Tape name (terminated with CR if less than characters)
22	Number of passes
24	ASCII description terminated with CR
104	Date and time when formatted Byte 104 - Days Byte 105 (Top 4 bits) - Year since 1981 Byte 105 (Bottom 4 bits) - Month Byte 106 - Hours Byte 107 - Minutes Byte 108 - Seconds
109	Reserved
128	

There then follow 1-14 entries of the form

0	Low 2 bits: 0-Blank 1-OK 2-Corrupt Bit 7 Set if non-MDFS disc
1	Disc name backed up
11	Date and time of backup Byte 11 - Days Byte 12 (Top 4 bits) - Year since 1981 Byte 12 (Bottom 4 bits) - Month Byte 13 - Hours Byte 14 - Minutes Byte 15 - Seconds
16	Data start block
19	Length in K
22	Error info
23	Reserved
64	

10.22 Password File Format

The password file, and the user information stored by the fileserver, is stored in the file %PASSWORDS. The following information is included for users who wish to write their own password handling programs eg password disclosing. It follows the following format :-

0	Entry number of 1st user entry with the first character less than 'A'
2	Entry number of 1st user entry with the first character as 'A' or 'B'
4	Entry number of 1st user entry with the first character as 'C' or 'D'
6	. . .
28	Entry number of 1st user entry with the first character greater than 'Z'
30	Entry number of the default user
32	Not used
64	1st User entry
128	2nd User entry
192	3rd User entry
256	. . .
n	Terminating User entry (Filled with &FF)
(n+64)	URD names and Libraries (pointed to by the user entry) Each a maximum of 80 characters terminated by a <CR>
z	

Each user entry (occuring in the password file every 64 bytes) has the following format :-

0	User Identifier (Terminated by a <CR> if less than 10 characters) Top bit must be masked out using AND &7F
10	Password (Terminated by a <CR> if less than 10 characters) Top bit must be masked out using AND &7F
20	Boot option
21	Flag bit 0 = Password unlocked bit 1 = System privileged bit 2 = No short SAVEs bit 3 = Permanent *ENABLE bit 4 = Acorn style Library bit 5 = Run only user bit 6 = Can't change auxiliary account
22	Offset from start of file to user's Root Directory Set to 0 if URD is normal
25	Offset from start of file to user's Lib. Directory Set to 0 if LIB is normal
28	Personal account number (0 = no personal account)
30	Bit map of high numbered accounts bit 0, &600-&63F bit 7, &7C0-&7FF
31	Reserved
32	Bit map of account ownership bit 0, byte 0=Account 0 bit 7, byte &1F=Account 255
64	

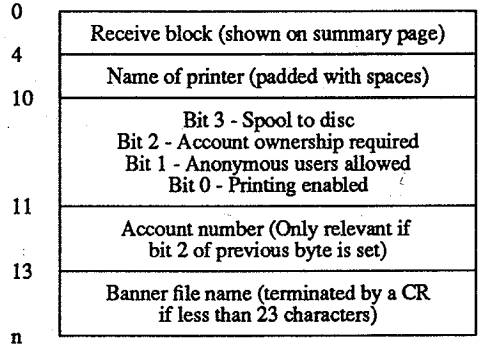
The top bits of the User Identifier and the Password hold a bit map of ownership to high numbered accounts.

bit 0, byte 0=Accounts &100-&13F
bit 7, byte 19=Accounts &5C0-&5FF

Setting the 'run only user' bit limits that user to calls 5,14,16,23,25,65, and 66, and commands *SDISC, *DIR, *LIB, *BYE and *I AM.

This information is not available on Acorn File Servers.

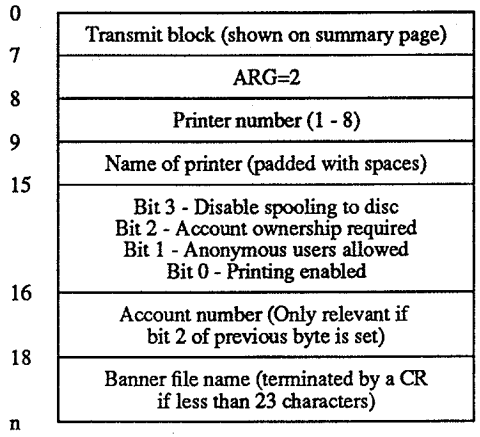
On exit,



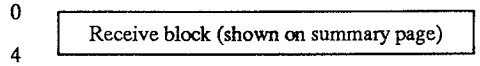
Write current state of printer

This call writes the detailed information about a printer, system privilege is required to do this.

On entry,



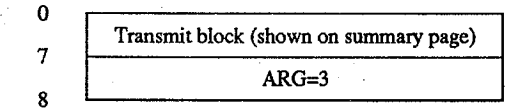
On exit,



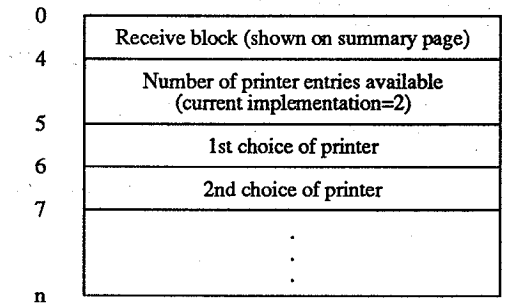
Read the AUTO printer priority

This call reads the order in which printers are selected for users who have not requested a particular printer.

On entry,



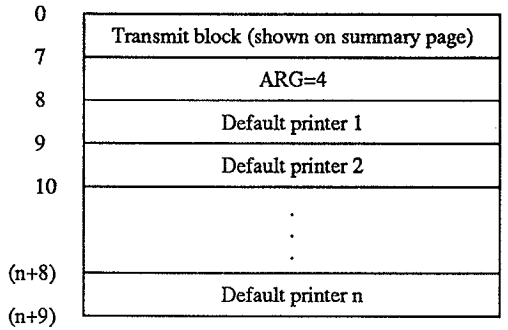
On exit,



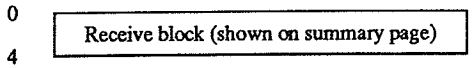
Write the AUTO printer priority

This call allows a privileged user to write the order in which printer are selected for users who have requested the AUTO printer.

On entry,



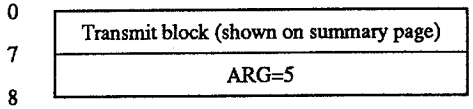
On exit,



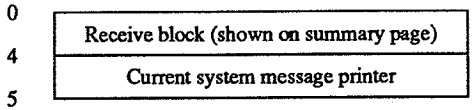
Read system message channel

This call returns the physical printer that all system messages are sent to. Note that the printer is a physical printer, so the parameter should be either 1 (parallel) or 2 (serial).

On entry,



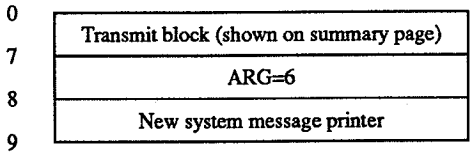
On exit,



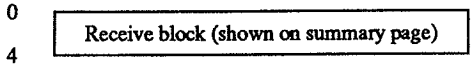
Write system message channel

This call allows a privileged user to set the physical printer that system messages come out of.

On entry,



On exit,

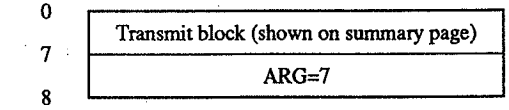


Read message level

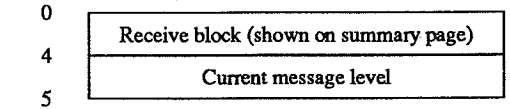
This call reads the current level of system messages. The value returned is in the range of 0 to 255. The amount of output is the level of output selected plus all the levels below that level. Therefore, in the list of levels shown to set the message level to 7 would make the file server print all logons and logoffs as well as errors.

Message level	Description
0	Off
5	Logon/logoff
7	Errors (i.e. 'Wrong password', 'bad name' etc.)
10	Maximum users and all star commands
11	Load/save
15	*cat and opens
128	Aborted loads
130	Function codes
150	Network errors
170	Map building names
200	Disc read/write
250	All successful network transactions to and from the fileserver
255	All Activity to the JPROC processor

On entry,



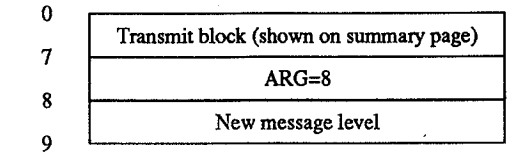
On exit,



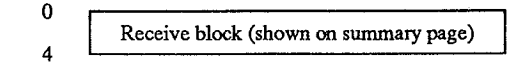
Set message level

This call sets the message level, as described above. It should never be neccessary to set the message level to greater than 127 and that setting the message level to a value greater than 150 produces excessive output and will probably reduce the performance of the file server.

On entry,



On exit,



Read the default printer

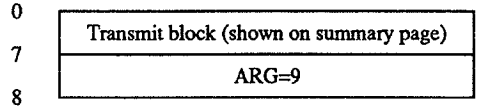
This call returns the default printer. This printer will be selected if a user starts to print without having selected a particular printer.

The values for the default printer are :-

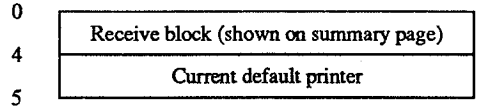
0 Automatic search through the list default printer priority list (set by Fn=65 ARG=4)
1 Logical printer 1
2 Logical printer 2
. .
. .
. .
8 Logical printer 8

255 Hold the job output in the %PRINTQ directory.

On entry,



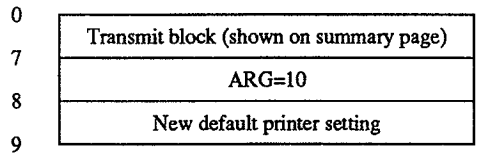
On exit,



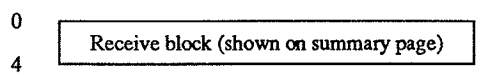
Set the default printer

This privileged call sets the default printer, see ARG=9 for more information about valid printer numbers.

On entry,



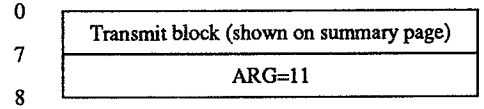
On exit,



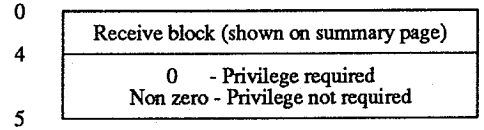
Read the privilege required to change time

This call reads the privilege required to change the file servers time. The SJ Research file server normally insists on system privilege to be able to change the time. However, if it is desired to change the time frequently this can be disabled.

On entry,



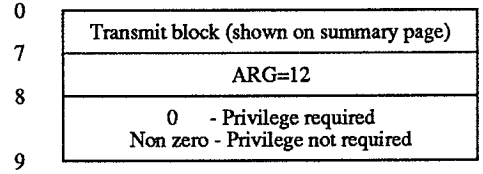
On exit,



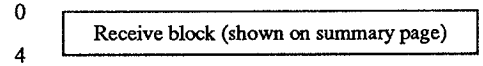
Set the privilege required to change time

This call sets the privilege required to change the file servers time.

On entry,



On exit,



N.B. These calls are only supported on the SJ Research file server.

10.22 Password File Format

The password file, and the user information stored by the fileserver, is stored in the file %PASSWORDS. The following information is included for users who wish to write their own password handling programs, such as implementing the Acorn File Server commands 'NEWUSER', 'REMUSER' and 'SPRIV'. It follows the following format :-

0	Entry number of 1st user entry with the first character less than 'A'
2	Entry number of 1st user entry with the first character as 'A' or 'B'
4	Entry number of 1st user entry with the first character as 'C' or 'D'
6	. . .
28	Entry number of 1st user entry with the first character greater than 'Z'
30	Entry number of the default user
32	Not used
64	1st User entry
128	2nd User entry
192	3rd User entry
256	. . .
n	Terminating User entry (Filled with &FF)
(n+64)	URD names and Libraries (pointed to by the user entry) Each a maximum of 80 characters terminated by a <CR>
z	

Each user entry (occuring in the password file every 64 bytes) has the following format :-

0	User Identifier (Terminated by a <CR> if less than 10 characters)
9	Password (Terminated by a <CR> if less than 10 characters)
19	Boot option
20	Flag bit 0 = Password unlocked bit 1 = System privileged bit 2 = No short SAVEs bit 3 = Permanent *ENABLE bit 4 = No Library bit 5 = Run only user
21	Offset from start of file to user's Root Directory Set to 0 if URD is normal
24	Offset from start of file to user's Lib. Directory Set to 0 if LIB is normal
27	Personal account number
29	Reserved
32	Bit map of account ownership bit 0, byte 0=Account 0 bit 7, byte &1F=Account 255
64	

This information is not available on Acorn File Servers.

10.22 Application Notes

Topics covered:

OSARGS
Printing
Econet Workspace
Use of OS workspace (or, How to make games work)
The program NETMON
Basic Guide to Econet Line Protocols

Glossary of terms and abbreviations used in this section:

AUG	Advanced User Guide for the BBC Microcomputer - Bray, Dickens & Holmes.
ESUG	The Econet System User Guide (The greyish book).
EAUG	Econet Advanced User Guide (The black book with sky and clouds on it).
UG	BBC Microcomputer User Guide - beware, details of OSFIND etc are WRONG.
OSHWM	Operating System High Water Mark -- A system variable, to which BASIC sets the value of PAGE.

The value of OSHWM depends on how many filing system ROMs are present in a BBC machine. (Languages do not take up any room, except when they are active). Here is a list of some common configurations:

ROMs fitted	value of OSHWM
No ROMs	&0E00
Disc	&1900
Econet	&1200
Disc & Net	&1B00
Teletext	&2200
Teletext & Disc	&2400
Teletext & Econet	&2400
Teletext, Disc & Net	&2600

File names

Although individual components af an Econet file name may not be longer than 10 characters, it is quite possible to have a compound file name (eg $.JOHN.BBCprogs.....2ndrev.OLD.developmnt.Addition) of almost infinite length. A compromise length of 80 characters should be used as a minimum. Even a DFS file name can be up to 12 characters long (:2.R.LONGEST), requiring 13 bytes of storage including the CHR$(13) on the end. Programs which disallow filenames over 7 characters long are highly unsatisfactory.

Econet workspace

When writing * command programs it is desirable to allow them to run in areas of memory which do not corrupt BASIC programs and other valuable user- or system data. The most obvious way of doing this with Econet-specific commands is to use the Econet workspace (Public area) which comprises pages &E and &F. When writing Filing-system independant commands use the area mentioned in note b) below.

As a general guide the following areas are safe:

E10..E1D	OK
E1E..E22	Corrupted during loading
E23..E2F	OK
E30	? DNFS stores file name
E31..EFF	OK
F00..F03	used by * commands, OSFILE etc.
F03..F04	must be zero
F05..F08	Corrupted during loading
F09..F0C	must contain the 32-bit execute address
F0D..FDC	OK

Notes:
a) F00 is used as a buffer for all FS-related commands eg *I AM , *<filename>, OSFIND etc.
b) It seems now to be an accepted convention that if you can't get * commands to work in the Econet workspace, then pages 9 and 10 (ie &900..&AFF) should be used.

Zero-page workspace

If you are writing your own * commands, and need some zero-page workspace, it is important to use the correct area. There is an area specifically reserved for such 'Operating System' commands, and this is at locations &A8..&AF. See AUG p. 268.

NB it is not good enough to use the 'spare' BASIC workspace &70..&8F, as there is no reason why a user may not run any * command from within any langauge, not necessarily BASIC.

Use of OS workspace

The area of memory below PAGE (or to be more precise OSHWM) is reserved for use by the OS and any filing systems that may be in your machine. Interfering with this memory can have unexpected and disastrous effects. However, many games and other (illegal, in the bad programming sense of the word) programs use RAM below PAGE, sometimes as low as &400 (an admittedly exceptional case).

On the Econet this is particularly nasty as the NFS uses NMI which will interrupt ALL machines whenever any net traffic occurs. This results in machines crashing when plugged into the network. In fact it is not only the NMI workspace which is corrupted during an interrupt but also the NFS Public workspace (pages &E and &F) and the NFS Private workspace, which effectively means any part of RAM below OSHWM.

The solution is for the user himself to 'claim' the NMI workspace, which has the effect of disabling the Econet altogether. It is best documented in the AUG pp. 320.

To claim NMI workspace use OSBYTE &8F, viz:

```
A%=&8F  (sideways ROM call)
X%=&C   (the 'claim' call)
Y%=&FF  (as it instructs you to do)
CALL &FFF4
```

or *FX143,12,255

Of course the main disadvantage of disabling the Econet is that you cannot use any of its facilities while the program in question is running. If you are writing your own software, disabling the Econet should only be considered as a last-ditch option. Educational users will be particulary annoyed as the majority have networks and may wish to use network related utilities with such software.

10.23 The Four-way Handshake

Data is transmitted across the network using a four way handshake. This transfer protocol is done automatically by the NFS ROM and the user need not be aware of its existence.

The first packet, sent by the sender machine, contains the destination station number, the source station number, the port on which the data is being sent, and the type of operation. This packet is called the 'scout packet'. The second packet, which is called the 'acknowledge packet', is sent by the destination station, if the following conditions are met :-

1) The machine is on the network.
2) The machine has a receive block open for that port.
3) The machine is not protected against the operation.

The third packet is the sender machine sending the data. The fourth packet is the destination machine's acknowledgement and means that the data has been correctly received.

All operations use this four way handshake with the exception of peek, machine type peek, halt, continue and broadcast. The four-way handshake can been seen using the utility 'NETMON', which is documented below.

Description of the Program NETMON

NETMON provides a means of continuously monitoring the network at a very low level. It displays the data bytes as they are sent down the network and also some status information, particularly useful in the debugging of network software, and networks in general.

**NETMON* loads some very special code which runs the network hardware directly. This effectively removes it from the network, as far as other machines and the program **STATIONS* is concerned. It is wise after running NETMON to power-off before attempting to use it for normal programming purposes.

The program prints:

```
ECONET MONITOR xxx
1E
```

where xxx is the station number of the monitor machine. Monitor output can at any time be stopped by pressing the space bar (ctrl-shift functions as normal).

Data bytes as they are sent on the network are printed in hex. There are various statuses that are printed and these are:

<space>
: Address present.
Indicates that the next byte is an address byte, which is always the first of a packet. Packets are therefore always separated by spaces.

i <newline>
: Idle detected.
Occurs between any two Econet messages. Stations wishing to transmit must always wait for an idle condition on the line (a sequence of 15 one's) before enabling their line drivers. A long string of i's is caused by network hardware problems, usually either poor wiring, or one or more blown SN75159(BBC)/26LS30(Master) line driver ICs.

v Frame valid.
Occurs just before the last byte of a packet, indicating that the CRC was valid.

e CRC error.
This can be caused by two stations transmitting simultaneously, or by noise getting into the network, or by an intermittent network connection, or a bad econet lead anywhere on the network, or....

o Data overrun error.
At clock speeds above 160KHz or thereabouts the monitor cannot keep up with the data rate. However, BBC machines are quite capable of running up to about 235Kbaud (2nd processors just under 200Kbaud reliably). Unless loss of data bytes, by the monitor program, is a nuisance, this should be of little concern.

b Abort. Normally an error, but at high clock speeds these can occur on a correctly functioning network (the **b** occurs in place of the **v**). One can also get packets like:

```
C800010080b i
99
```

where it would appear that an idle has occured in the middle of a packet! This is due to the receive FIFO register in the SDLC chip, which buffers the data bytes but not the statuses. In fact in real time the idle occured after the data bytes.

d Clock missing.
Lots of **d**'s indicates that the clock (to the monitor station at least) is intermittent. Suspect Econet lead, or clock connection in the network, or clock itself.

A Basic Guide to Econet Protocols.

Hex numbers are preceded by &, all other numbers are in decimal.

A standard Econet interchange might look like this:

```
FE00120080v99 1200FEv00 FE00120090030101020300OBv0D 1200FEv00 i
```

which is one of the messages sent when doing a *CAT.

Each of the groups of numbers separated by a space is called a *Packet*, and a group of packets constituting one of the legal Econet transfer protocols is called a *Message*. The packet is the basic unit on the Econet. Packets consist of:

One or more flags (not displayed on monitor)
Any number of data bytes
2-byte CRC (only correctness or incorrectness displayed on monitor)
One flag (not displayed on monitor)

Data bytes within the packet follow with no gaps and there are special encoding techniques which ensure that the flag pattern does not occur within a packet. The packet structure is known as SDLC, and it is constructed and decoded in hardware by a chip in the Econet circuit. On the BBC machine an MC68B54 is used (IC 89) and on Z80-based machines a Z80 SIO.

You will notice that each packet has the same 4 bytes on the front but the order changed. All Econet packets have a 4-byte header describing where the message is going, and who sent it. Both these 'addresses' are 2-byte quantities, the first byte being a station number and the second a gateway number. Normally the gateway number is zero, unless you have a *Bridge* on your network in which case this may be non-zero. A zero gateway number addresses your 'local' network.

The first 'address' is the *destination* address. This is on the front of a packet so that any machine (all machines listen all the time) can tell whether that packet is directed at it or at another machine. The second address is the *source* address, ie the network address of the machine which transmitted the packet. In the first packet **FE00** was the destination address and **1200** was the source address.

You should see now that, in the example above, 2 packets were going from station &12 to station &FE, and two from station &FE to station &12.

Let us now examine the message in more detail, packet by packet.

The first packet is a *scout packet*, sent to station &FE (=254, so probably the FS) from station &12 (a client). In addition to the packet header, there are two bytes; the first is a *control* byte, the second the *port* number identifying the rest of the message. (NB the word port here is nothing to do with hardware ports, although it has much the same function as an identifier.) The control byte isn't very important as regards the FS interface: it can be used for sequencing and is used extensively for printing.

The port number is much more interesting. Stations which are set up to receive messages can selectively allow messages from <all stations or one particular station> and on <all ports or a specified port>. The port number identifies the data to the receiving station as 'this is the data I wish to SAVE' or 'here are some bytes to be printed' or 'this is a print status enquiry': &99 has identified the message as an *FS command.*

The second packet is an acknowledgement. The first packet was sent from the client to the FS. The acknowledgement packet goes the other way (notice the header bytes are the other way around) and it is a

packet telling station &12 that station &FE has recognised the scout package and is prepared to receive some data.

The third packet is a data packet. After the header there follow data bytes. The protocol does not specify in advance how many bytes are going to be sent, and it is up to the transmitter (station &12) to decide how many he is going to send. If too many are sent an error will be reported at both ends. Here are the data bytes again (less the packet header):

```
90 03 01 01 02     03 00 0B 0D
```

To interpret these bytes we first have to remember the port number in the scout packet, which was &99, the FS command port. This defines the first 5 bytes of data to be a *Standard Tx Header* (See page 63 of EAUG, page 99 of ESUG), so that the reply port is &90, Function code is 3 (an 'Examine' call, used by *CAT) and context handles are 01, 01 and 02.

The rest of the data bytes are parameters to the examine call (p 65 of EAUG, p.105 of ESUG), which gives ARG=3, and requests &0B entries starting from entry 0 in directory "" (ie the CSD). An ARG of 3 tells the FS that the data returned should be file title + access, in ASCII.

The fourth packet is also an acknowledgement. It tells stations &12 that stations &FE has received the data correctly, and that not too much data was sent.

We have looked at a successful Econet transfer. It is also wise to know about transfers that didn't work; the most likely of which is when the error reported is **Not listening**. This can happen in various ways; either the distant machine is not present, or switched off, or it has not been set up for receive or it may have crashed. In either case the monitor output will look like this:

```
FE00120080v99 i
FE00120080v99 i
FE00120080v99 i
FE00120080v99 i
FE00120080v99 i
FE00120080v99 i
```

ie station &12 repeatedly sending scout packets to station &FE but getting no acknowledgement (It sends a few hundred such packets before reporting an error).

This sort of monitor output may also occur under perfectly normal circumstances: if an FS is busy with another client and/or there is disc activty going on it will not be set up to receive packets on its command port, and so will not send any acknowledge packets.

A less likely form of error may look like this:

```
 FE00120082vD1 1200FEv00 FE001200000D0A03v31 i
```

(again many times). Station &FE has acknowledged the scout packet from station &12, but has not acknowledged the data packet. Almost certainly too many bytes were sent (the BBC machine will report **Net error**)