Acorn Econet

Advanced user guide

Within this publication the “BBC” is used as an abbreviation for "British Broadcasting Corporation"

412,010 Issue 1
October 1983

Written and designed by Baddeley Associates, Cambridge.

All correspondence should be addressed to:

Technical Enquiries
Acorn Computers Limited
Fulbourn Road
Cherry Hinton
Cambridge
CB1 4JN

Contents

Econet for advanced users

This guide deals with features of Econet and its filing system that are not covered in the other Econet user guides. Many of these features are available only on Level 2 systems, and the guide has been written with Level 2 users primarily in mind. The first two sections describe:

- how to use the network filing system from programs
- how to use the network itself from programs.

These sections introduce the facilities available and include example programs and program lines. Later in the guide, there is a reference section which gives all the details you need to use these facilities, once you have become familiar with the principles.

After the two programming sections, there are sections which describe:

- how, in detail, communications between stations are handled by the network
- a set of additional Econet commands, not covered in other Econet user guides, which enable you to communicate with, or take over, other stations.

At the end of the guide, you will find a note on Econet error handling, two circuit diagrams and an index.

You will need to have read:

- the *BBC Microcomputer* User Guide, which we refer to as the BBC *User Guide*
- the Econet *Level 2 File* Server user guide, which we refer to as the Econet user guide.

The examples in this manual are of BASIC programs, but the principles are the same for assembler programs.

Conventions used in this guide

The following abbreviations are used for the four processor registers:

A	accumulator
X	X register
Y	Y register
C	carry bit of the processor status register.

When X and Y hold a two-byte value, X holds the least-significant byte, and Y the most-significant byte.

D, **N**, **V** and **Z** are used for flags in the processor status register.

& indicates a hexadecimal number.

Multiple-byte values are held in 6502 order: least-significant byte at the lowest address and most-significant byte at the highest address.

Buffer addresses are four bytes long, to accommodate the addressing range of 16-bit and 32-bit second processors.

Control blocks are shown with addresses increasing down the page. The offset of the first byte of a field is marked opposite that field, with the length of the last field (where it is fixed) indicated by an offset for the next unused location.

Programming with the filing system

The Econet filing system is only one of the several filing systems available for BBC Microcomputers – others include the cassette system, the local disc filing system and the telesoft system.

All these systems can be used from your own programs, and the methods you use are in principle the same for all of them. But there are some variations. This section gives you all the information you need to program with the Econet filing system.

There are three ways you can use the Econet filing system from your programs:

- including file server commands in your programs
- using the BASIC file-handling keywords, like BGET# and PTR#
- calling operating system routines directly.

Including file server commands in programs

This is the simpler of the three methods.

EXAMPLE

```
100 *LOAD DATA 3000
```

This program line would load the file DATA into memory at. *address 8c3000.*

In a line like this, the characters after the + are sent direct to the operating system command line interpreter (OSCLI) routine, so BASIC variables cannot be included in the line.
In BASIC II, there is an OSCLI function which takes a single string as its argument and sends it to the command line interpreter. This function enables you to include BASIC variables.

EXAMPLE

```
10 filename$=" DATA"
20 OSCLI "LOAD "+filename$+ " 3000"
```

In both versions of BASIC, you can call OSCLI directly, using a routine which is part of your machine’s operating system: this is described in the *BBC User Guide.*

The distinction between BASIC I and BASIC II, and the way to find out which version you have, are explained towards the back of the Econet user guide.

Using BASIC file-handling keywords

This method is described in the *Econet* user *guide.*

Using operating system routines

The third way of using the Econet filing system from your programs is to call operating system routines directly.

In the BBC Microcomputer, 16K of ROM contains the operating system (OS). Within this area there is a set of routines, which you can use by giving their entry addresses. When you include a file server command in a program, the operating system will interpret the command and then call the appropriate operating system routine. You can also call a routine directly: this is more complicated, but gives you a wider range of activities. You can, for example:

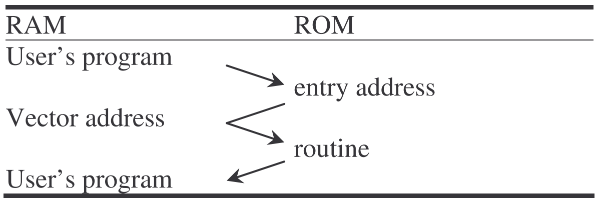
- change a file’s reload address, execution address and attributes
- read the entries in a catalogue
- find the length of a file.

There are two BASIC keywords which you can use to invoke an operating system routine: CALL and USR. CALL is used to run the routine, USR to run the routine and put its result into a BASIC variable. These two keywords are explained in detail in the BBC *User Guide.*

The account of operating system routines that follows assumes that you are familiar with CALL, USR, and the use of A%, X%, Y% and C%.

Each routine has a vector address in RAM. The entry address you give contains an instruction to:
- find the vector address
- look at the ROM address held there
- go to that ROM address.

In other words, the routine is reached by jumping through the vector address. The contents of the vector address in RAM may be changed to allow for different ROM arrangements in different operating systems.



There are six operating system routines available to programmers who want to make this sort of direct use of the Econet filing system:

- OSFILE
- OSFIND
- OSARGS
- OSBPUT
- OSBGET
- OSGBPB.

Some of these routines carry out one particular function, but most can carry out a range of related functions. OSFILE, for example, carries out a range of functions related to reading and writing file or catalogue information, including:
- loading a file
- saving a file
- writing a file's catalogue information
- deleting a file or directory.

The reference section at the back of this guide gives complete details of the functions of all six routines.

To use any of these routines, you:

1 reserve memory for the items of information the routine needs
2 fill in the reserved memory with that information
3 assign values that will be transferred to some or all of the registers A, X and Y (A, for example, is often used to select a particular function from the routine's range of functions)
4 call the routine, using the BASIC keywords CALL or USR, and giving the entry address of the routine
5 evaluate the results.

Reserving memory

This is done using the DIM statement. For example

10 DIM file% 30

DIM here reserves a series of bytes nationally numbered 0 to 30 – that is, 31 bytes – for the variable file%, and puts the address of the bytes into file%.

Filling the memory

To fill the reserved memory with the information needed by the routine, you use the indirection operators:

? the byte indirection operator
! the word indirection operator
$ the string indirection operator.

These operators are explained in the *BBC User Guide,* chapter 39.

EXAMPLES

100 ?M%=0

means: set *the contents of the byte of* memory reserved *at location M% to 0.*

110 PRINT !M%

means: print in hex the contents *of* the *four* bytes *starting* at *M%*.

120 $M%="ECONET"

means: put the ASCII codes *representing the string ECONET into* memory, *starting at location M%,* and *ending the string with a carriage* return.

Assigning entry values

In this stage, you assign values to the BASIC variables that CALL or USR will look at. The CALL or USR function will transfer these values into the four registers.

Variable		register
A%	→	accumulator (A)
C%	→	the carry bit (C) of the processor status register (P)
X%	→	X register (X)
Y%	→	Y register (Y)

EXAMPLE

120 A%=&40

This line puts the hexadecimal number 40 *into the variable A%. The CALL or USR function will transfer this value to the A* register. *This register is normally used to* select a *particular function from the range of functions available from a routine.*

All BASIC variables contain four bytes, but each register can take only one byte, and C takes just one bit. CALL and USR handle this by transferring only:

- the lowest byte of A%, X% or Y% to A, X or Y
- the lowest bit of C% to C.

EXAMPLES

A%	A
&0000F3FE	&FE
&3423CB22	&22

C%	C
&00033224	binary 0
&000000F1	binary 1

Often, you will want to give the routine two-byte values, which you will have to split between two registers.

You do this with the calculation DIV 256, which has the effect of removing the lowest byte – that is, the last two hexadecimal digits.

EXAMPLES

23,768 *is a two-byte value: in hex,* &5CD7.

23,768 *divided by 256 is 92, remainder 216.*

The DIV function performs this calculation, ignoring the remainder. So 23,768 DIV 256 is 92: in hex, &5C.

The effect, then, of the DIV calculation is to remove the last two hex digits:

&5CD 7 – > &5C

To split 23,768 between the *two* registers X and Y:
- set X% to &5CD8
- it will contain, in full, &00005CD8
- set Y% to (X% DIV 256)
- Y% will contain &0000005C
- only the last byte of X% – &D8 – will be transferred to X
- only the last byte of Y% – &5C – will be transferred to Y.

$$\begin{tikzcd}[ampersand replacement=\&]
\text{\&5CD8} \arrow[dr] \& \& \\
\& \text{X\%: }\underline{\text{\&00005C}}\text{D8} \arrow[d] \arrow[ddr] \& \\
\& \text{Y\%: \&000000}\underline{\text{5C}} \arrow[d] \& \\
\& \text{Y: \&5C} \& \text{X: \&D8}
\end{tikzcd}$$

```
130 X%=f ile%
140 Y%=X% DIV 256
```

This example splits the contents of the variable file% between X% and Y%.

When you call the routine, the value in X%, the least-significant byte, will be put into the X register, and the most-significant byte (the value in Y%) will go into the Y register.

Calling the routine

You now call the routine, using CALL or USR.

EXAMPLES

```
200 CALL &FFDD
```

This calls the routine whose entry address is hex FFDD.

```
210 U%=USR &FFCE
```

This calls the routine at &FFCE, putting the result into the variable U%.

Evaluating the results

The simplest way to evaluate the results of a routine is to use USR, and print the contents of the variable into which USR puts its result.

EXAMPLE

```
220 PRINT ~U%
```

This means: print in hex the number in U%.

The result from a USR call will be a four-byte variable, containing the values in the registers at the end of the routine; in hex it will look something like this:

$$\begin{tikzcd} \underline{71} \arrow[d] & \underline{1B} \arrow[d] & \underline{84} \arrow[d] & \underline{24} \arrow[d] \\ P & Y & X & A \end{tikzcd}$$

You are likely to be interested in only part of this result – just the contents of the X register, perhaps. To find the part of the result you want, you mask off the other sections of the variable.

EXAMPLES

```
10 P%=U% AND &FF
20 PRINT ~P%
```

displays &24, the contents of the A register. (The BBC User Guide *explains how to use AND.)*

```
10 P%=U% AND &FF00
20 Q%=P% DIV 256
30 PRINT ~Q%
```

displays &84, the contents *of the* X register.

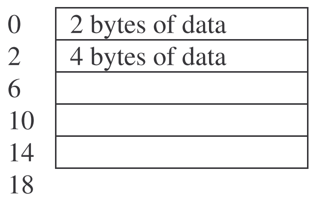
Using a routine to open a file

This program uses the routine OSFIND to open the file DATA for input only.

100 DIM file% 30	reserve 31 bytes for file%
110 $file%="DATA"	put the address of the filename DATA into the variable file%
120 A%=&40	select the function in OSFIND that opens a file for input only
130 X%=file% 140 Y%=file% DIV 256	split the address of the filename DATA between the X and Y registers
150 U%=USR &FFCE	call OSFIND and put the result in the variable U%
160 ch%=U% AND &FF	mask off U% so that the channel number (in the A register) is shown
170 PRINT ch%	display the channel number

Control blocks

Two of these six routines, OSFILE and OSGBPB, make use of control blocks. These are small areas of memory set aside to hold the data that you want the routine to handle. Within a block, data is arranged in fields of varying sizes. The arrangement of control blocks varies with different routines and operations. A typical 18-byte block might be arranged like this:

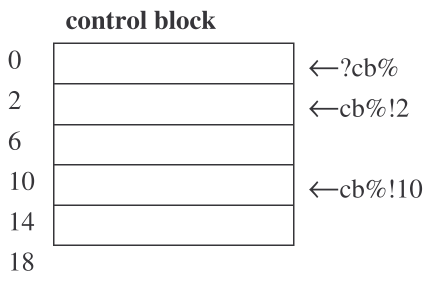


The numbers at the left of the diagram are the addresses in decimal of each field, given as an offset from the location of the start of the block. Each field will hold a particular piece of data. The reference section gives details, for each routine, of what kind of data to put in each field.

When you use OSFILE or OSGBPB, you reserve memory for the block, fill the block, and give the routine pointers to the block.

Filling a control block

To fill the control block, you use the indirection operators. In this example, the control block starts at location cb%.



Pointing to a control block

To point to the control block, you split the start address of the block – the variable cb% – between the X and Y registers.

Using a control block

The following example changes the reload address of the file DATA to %6000, using the routine OSFILE.

10 DIM cb% 20 20 DIM file% 30	set aside memory for a control block (cb%) and a string file%
30 $file%="DATA"	put the filename DATA into memory at the value of the variable file%
40 !cb%=file% 50 cb%!2=&6000	fill the control block with the data it needs
60 X%=cb% 70 Y%=cb% DIV 256	point to the control block, by splitting the contents of the variable cb% between the variables X% and Y%, for transfer to the X and Y registers
80 A%=2	select function 2, which changes the reload address
90 CALL &FFDD	call OSFILE, whose entry address is &FFDD

The routine then executes, using the control block you have set up.

Starting work

You can now start programming with the Econet filing system, using the six operating system routines we have mentioned. You will find details of all their addresses, functions, control blocks and results in the reference section at the back of this guide.

Programming with the network

This section tells you how to write programs that send data between Econet stations. There are two kinds of network communication: co-operative and immediate. Co-operative operations require the participation of two stations: you will need to run a TRANSMIT program at one station and a RECEIVE program at the other. Immediate operations enable you to communicate with another station just by running one program at your station.

Both types of operation are controlled by software in the Econet EPROM. To use them, you need not have Econet as your selected filing system (you could be working under the disc filing system, for example).

Like filing system programs, network programs make use of operating system routines. The two you will use are:

- OSWORD
- OSBYTE.

OSWORD works with a control block. Routines and control blocks are explained in the previous section, *Programming with the filing* system.

In this section, we work through an example of a co-operative operation, which enables you to use your keyboard to play musical notes at another station. This operation uses a TRANSMIT program and a RECEIVE program: there are full listings of both programs at the back of the guide.

There is a third kind of co-operative operation, BROADCAST, which is described in the reference section.

Transmitting and receiving

When you send data through the network, you specify:

- a station number
- a port number.

In network programming, station numbers are two-byte values. The most-significant byte is usually zero, and is used only in installations where several networks are connected together. The numbers you can use are & 01 to &FE. Stations &00 and &FF are special cases. In transmitting operations, they are reserved for BROADCAST; in a RECEIVE program, you can specify station &00, in which case your station will accept messages from any other station.

Port numbers allow stations to carry on more than one conversation at a time. You specify a particular port at the receiving station, on which it will accept your message. The station will be able to receive other messages at the same time:

– from your station, on other ports
– from other stations, on the same port.

The range &01 to &FF is available. For your TRANSMIT and RECEIVE programs to work, the receiving port number must match the transmitting one. You can set up a station to receive on port &00, in which case it will accept every message sent it, whichever port numbers those transmissions gave.

Stations &EB and &FE are normally the printer server and the file server. Ports &90 to &9F, &DO and &D1 are reserved.

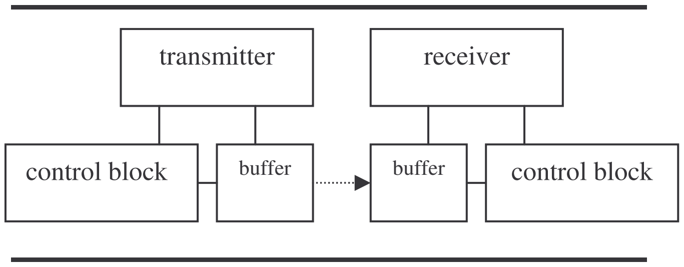
The TRANSMIT and RECEIVE programs each use:

- a control block
- a data buffer.

Each control block contains details of:

- the type and status of the operation
- the number of the station transmitting or receiving
- the port number
- the address of the data buffer.

The transmitting station's data buffer is the area of memory that will contain the data to be transmitted; the receiving station's buffer is the area where that data will be stored, once it has been received.



In your TRANSMIT program, you:

- set up your TRANSMIT control block and your buffer
- fill the control block
- put the data you want to send into the buffer
- call OSWORD to start the transmission, pointing to your control block.

The NFS software will take a copy of the TRANSMIT control block, and put the copy in its own memory. The data in that copy will change, depending on the status of the transmission – so you can find out what stage the transmission has reached by examining the copy. You do this by:

- calling OSBYTE, to check that the transmission has finished, and whether or not it has succeeded.

In the RECEIVE program, you:

- set up a buffer into which you will receive data, and a RECEIVE control block
- fill the control block
- call OSWORD to open the RECEIVE block, so that your station is ready to receive data.

Again, the NFS software will make a copy of your block. The OSWORD call will give you a number by which you will be able to refer to that copy, so that you can check the progress of the reception. You do this by:

- calling OSBYTE, giving the RECEIVE control block number.

When some data has arrived, you:

- call OSWORD again to read and delete the control block copy
- read the data received.

The control blocks you use look like this:

	TRANSMIT	
0	control byte	must be set when the routine starts to any value, like &80, in which the top bit is set: will be reset to 0 if the transmission fails to start
1	destination port	the number of the port you want to transmit to (1 byte)
2	destination station	the number of the station you want to transmit to (2 bytes)
4	address of buffer start	(4 bytes)
8	address of buffer end	(4 bytes)
12		

	RECEIVE	
0	&00	
1	flag	indicates whether the block is ready to receive; should be set to &7F when the routine starts; when a transmission is received, the TRANSMIT control byte is written here
2	receiving port	(1 byte)
3	station	(2 bytes)
5	address of buffer start	(4 bytes)
9	address of buffer end	(4 bytes)
13		

The addresses of the starts and ends of the buffers are four-byte values, so that 16-bit and 32-bit second processors can be used.

The operating system has room for up to 16 control block copies. Reading a block back using OSWORD has the effect of deleting the copy; you can also use OSBYTE to delete a control block copy without reading it back.

When you have written the TRANSMIT and RECEIVE programs, you run them simultaneously at the two stations you want to use. It is vital to make sure that the two programs agree on which station numbers you are using, and which port you want to receive at.

Checking for errors

Writing programs to communicate between two machines is more complicated than writing for a single machine. This is partly because there are several ways in which an attempt to transfer data could fail, and you need to be able to identify what went wrong and take appropriate action.

In particular, a transmission may fail to start because:

- another transmission is in progress from the transmitting station
- the transmitting station's disc drive is in use: that is, the disc drive is actually spinning.

A transmission may start but fail to finish because:

- your TRANSMIT and RECEIVE control blocks fail to match
- you are trying to send more data than the RECEIVE block is set up to accept
- there's something wrong with the network (your station is not plugged in, the clock or the terminators are not switched on, or there's a hardware fault)
- a packet of data has been damaged by electrical interference (this happens very rarely, but your programs should allow for the possibility).

In your program, you will need to distinguish between three possible results:
– success
– fatal error
– non-fatal error.

In the first case, data will successfully have been transmitted. In the second, there is something wrong with the network or the control blocks, so that there is no point in trying to transmit again. In either case, you will want the program to continue, and not repeat the transmission.

The third category, non-fatal error, covers cases where the transmission may have failed for a temporary reason:
– electrical interference
– the remote station is busy, and therefore does not
– for the moment have an open RECEIVE control block
– the network is at that moment busy.

In these cases it's worth trying to transmit several times before finally deciding that the operation has failed – most likely because the station you are trying to transmit to is not plugged in, or does not have a RECEIVE block set up.

If an error has occurred, the OSBYTE call used to check a transmission's progress will return one of these values:

&40	network jammed	fatal
&41	packet damaged or RECEIVE buffer too small	non-fatal
&42	receiving station not listening (perhaps because the control blocks don't match)	non-fatal
&43	not plugged in, or clock not switched on	fatal
&44	Program error: a badly-formed TRANSMIT control block	fatal

Constructing a TRANSMIT program

1 Reserve memory for your control block and buffer.

```
70 max_tx_length%=20
90 DIM cblock%13,txbuffer% max_tx_length%
```

Line 90 reserves memory for your TRANSMIT control block and your buffer. Line 70 sets the variable max_tx_length% to the maximum amount of data you expect to transmit.

2 Fill the control block.

```
310 cblock%?1=port%
320 cblock%!2=station%
330 cblock%!4= txbuffer%
340 cblock%!8= txbuffer%+length%
430 ?cblock%=&80
```

The values of port%, station%, txbuffer% and length% would be given elsewhere in your program.

3 Say what will be transmitted.

```
140 key%=GET
150 PRINT CHR$key%;
160 ?txbuffer%=key%
```

These lines print and put into the buffer a character representing the key you press at the transmitting station.

4 Give the OSWORD routine its entry values.

```
410 X%=cblock%:Y%=cblock% DIV 256:A%=&10
```

This sets the X and Y registers to point to your control block, and the A register to function &10, OSWORD's TRANSMIT function.

5 Call OSWORD.

```
60 OSWORD=&FFFl:OSBYTE=&FFF4
(program lines)
440 CALL OSWORD
```

Line 60 puts the entry addresses of each of the routines you will use into variables; 440 then calls OSWORD.

6 Make sure your transmission has been able to start. When a transmission fails to start, the control byte is set to 0; so you read your TRANSMIT control block, looking at the control byte. If it's 0, try to start transmitting again. Otherwise, go on to the next stage of the program.

```
360 REPEAT
(lines starting the transmission)
450 UNTIL ?cblock%<>0
```

Inside this loop is a line that resets the control byte to the correct value each time:

```
430 ?cblock%=&80
```

7 Check whether your transmission has finished. The process is called polling.

```
490 REPEAT
500 A%=&32:U%=USR OSBYTE
510 UNTIL (U% AND &8000)=0
```

These lines repeatedly examine the status of the transmission. Line 500 sets the A register to &32, the "poll transmit" function, calls OSBYTE, and has the result put into the variable U%.

When a transmission is completed, the top bit of the X register is set to 0. Line 510 masks off this element of U% by ANDing U% with the binary value 1000000000000000 – in hex, &8000.

8 Check the result of the transmission.

```
350 tries%=10:delay%=50
360 REPEAT
(lines setting up and starting transmission)
550 txresult%=(U% AND &FF00) DIV 256
555 nonfatal%=txresult%=&41 OR txresult%=&42
560 IF nonfatal% THEN
PROCdelay(delay%):tries%=tries%-l
570 UNTIL tries%=0 OR NOT nonfatal%
```

Line 550 masks off the X register byte in the four-byte value of U%, and puts the contents of X into the variable txresult%. The other lines mean:

– if the value in X is &41 or &42 (that is, non-fatal), delay (using a procedure defined elsewhere in the program), and try again, up to ten times.

9 Check what error occurred, if any. You do this by including lines which:

– examine txresult%
– print out something like “Success” if txresult% is 0
– otherwise, print an appropriate error message, depending on the value of txresult%
– end the program.

Constructing a RECEIVE program

1 Reserve memory for your control block and data buffer.

```
40 max_rx_length%=20
50 DIM cblock%13,rxbuffer% max_rx_length%
```

2 Fill the control block.

```
215 ?cblock%=0
220 cblock%?l=&7F
230 cblock%?2=port%
240 cblock%!3=station%
250 cblock%!5=rxbuffer%
260 cblock%!9=rxbuffer%+max_rx_length%
```

3 Give OSWORD its entry values.

```
300 X%=cblock%:Y%=cblock% DIV 256:A%=&11
```

4 Call OSWORD.

```
30 OS WORD=&FFF1: OSBYTE=&FFF4
310 CALL OSWORD
320 rxcb_number%=?cblock%
```

OSWORD takes a copy of your control block, and gives you a number by which you will refer to the copy; line 320 puts this number into the variable rxcb_number%.

Your machine is now waiting, ready to receive data.

5 Call OSBYTE to poll the RECEIVE control block.

```
360 A%=&33:X%=rxcb_number%
370 REPEAT:U%=USR OSBYTE
380 UNTIL (U% AND &8000)<>0
```

These lines repeatedly call OSBYTE, sending the routine:
– the function number &33
– the control block number.

The result of the routine is returned in the X register, and is the control byte of the NFS copy of your RECEIVE block. This byte will change, when the buffer receives data, from &7F to some value with the top bit set. The OSBYTE call repeats itself until the top bit of the X register changes to 1.

Once that has occurred, the data is in the buffer.

6 Use OSWORD to check the result of the reception, by reading back the network copy of the RECEIVE control block.

```
420 X%=cblock%:Y%=cblock% DIV 256:A%=&11
430 ?cblock%=rxcb_number%:CALL OSWORD
```

These lines call OSWORD, sending the routine:
– the function number &11
– a pointer to the area you want to copy the control block into
– the RECEIVE control block number.

The routine copies the control block, altering it to contain:

– the number of the station that sent the data (if the original control block gives 0 for the station number)
– the number of the port the data was received on (again, if the original had 0)
– the new value of the "end buffer" pointer (this value changes depending on the amount of data received into the buffer).

7 Find out how much data was actually received.

```
440 bytes_received%=cblock%!9-cblock%!5
```

This line subtracts the value of the "end buffer" pointer from the value of the "start buffer" pointer.

8 Read the data.

```
110 key%=?rxbuffer%
120 PROCSOUND(key%)
```

Line 120 invokes a procedure, defined elsewhere in the program, that will translate the key data received into sound.

Immediate operations

Immediate operations are those which you can carry out without the co-operation of another station: you do not need to set up any program at the other station. One immediate operation, for example, enables you to read an area of another station's memory.

To carry out an immediate operation, follow the steps you go through in a TRANSMIT program. The control block you use will be different with different operations; the details you need are in the reference section.

You can protect your station against another station's immediate operations with an OSWORD call. Again, details are in the reference section.

Other OSWORD and OSBYTE calls

There are several other useful OSWORD and OSBYTE calls, which you can use to program with the network.

call	**A set to**	**first byte of control block set to**	**function**
OSWORD	&13	function code	reading and writing station information
OSWORD	&14	&00	communicating with the file server
OSWORD	&14	&01	sending text messages
OSWORD	&14	&02	causing a remote error
OSBYTE	&35		severing a remote connection

These calls are described in detail in the reference section.

Frames and the four-way handshake

This section describes what happens during a TRANSMIT operation. The sequence of events is determined by software which drives a controller chip – the MC68B54 Advanced Data Link controller – in the Econet interface in your BBC Microcomputer. The 68B54 sends data in bundles called packets or frames.

During a single Econet TRANSMIT operation, four frames are exchanged. The sequence is called a four-way handshake:

1 the source station sends out an “Are you there‘?” frame, called a scout

2 the destination station sends back an acknowledgement

3 the source station sends its data

4 the destination station returns a final acknowledgment.

During a four-way handshake, no-one else can use the network.

The handshake ensures that a transmitter can tell for certain whether its message has been successfully received: if the final acknowledgement arrives, the data must have reached its destination.

The four-way handshake

The scout frame is ten bytes long, and is arranged like this:

1 byte flag	2 bytes destina-tion station	2 bytes source station	1 byte control byte	1 byte port byte	2 bytes CRC	1 byte flag

The opening and closing flags mark the beginning and end of the frame, and always contain 01111110. Information about the destination and source stations is held in each case as a pair of bytes: the first of the pair contains the station number itself. The CRC is a 16-bit cyclic redundancy check, the result of a calculation carried out on all the bits (except the flags and any inserted zeros) in the frame.

The destination station now compares the scout with its open RECEIVE control blocks, to try to find a match. While the search takes place, it sends out a stream of flags to prevent other stations from claiming the line. The process is called flag fill.

If it finds a match, it stops flag fill and sends an acknowledgement in a frame eight bytes long, arranged like this:

1 byte	2 bytes	2 bytes	2 bytes	1 byte
flag	destina-tion station	source station	CRC	flag

When it receives this acknowledgement frame, the transmitting station begins to send its data. The length of the data frame depends on the size of the data; its arrangement is like this:

1 byte	2 bytes	2 bytes		2 bytes	1 byte
flag	destina-tion station	source station	data	CRC	flag

Finally, as long as the transfer was successful, the receiving station sends its second acknowledgement, identical in form to the first.

If the final acknowledgement does not get through, the transmitter cannot be sure whether the data was received or not. In some applications, it may be risky to re-transmit, because the receiving station might, at the end of the process, have two sets of identical data. This problem is characteristic of the four-way handshake protocol. The solution is to identify each transmission with a number, either in the data itself or in the control byte, so that the receiver can distinguish between a retransmission and a new transmission.

The monitor utility

Every Econet system has on its master disc a utility program called NETMON (in Level 1 File Servers) or NETMONITOR (in Level Z). This program enables you to see frames on the screen – making it easy, for example, to find out what has gone wrong when a transmission fails.

Type: ***NETMON[RETURN]**
on a Level 1 system

or: ***NETMONITOR[RETURN]**
on a Level 2 system

A message like this will appear:

Screen: **ECONET MONITOR 001**
1E

From now on, every communication on the network will show up on your screen.

EXAMPLE

In this example, a *user at station 189 *DELETEs a* file.

```
FE00BD0080v99   BD00FEv00   FE00BD0090000102044454C455445v0D   BD00FEv00 i
BD00FE0080v90   FE00BDv00   BD00FE0000v00   FEBDv00 i
```

scout   acknowledgement   data   acknowledgement

The process involves two four-way handshakes. The four frames in each handshake are arranged horizontally: scout, acknowledgement, data, acknowledgement. In each frame, the flags and the CRC are not shown. The "i" at the end of each line shows that the network is, at that point, idle.

From this screen display, you can see immediately the contents of each frame. For example, the first scout shown contains:

1 byte	2 bytes	2 bytes	1 byte	1 byte	2 bytes	1 byte
flag	destina-tion station	source station	control byte	port byte	CRC	flag
	FE 00	BD 00	80	99		

FE is the file server number, 254 in decimal, and BD is the station number, 189. The “v” character shows that the frame has been checked, using the CRC, and is valid.

To make the screen easy to read, the program:
– shows a space between each frame
– starts a new line after every “i”
– truncates long data frames.

The monitor facility is useful for checking up on transmission failures. This screen, for example, shows a scout frame that has received no acknowledgement:

```
FE00BD0080v99 i
```

The result of this would be a &42 error.

In this example, station 189 has tried to send station 1more data than its buffer was set up to receive:

```
0100BD0080v99 BD0001v00 0100BD00AABBCCb i
```

The four-way handshake has been aborted in the middle of the data packet. The “b” indicates an aborted handshake.

Apart from i, v and b, three other status letters can appear:

e	CRC check has failed, because of noise on the network or electrical interference
d	the data carrier detect input to the 68B54 has changed: probably, no clock signal is being received
o	overrun: the network is running too fast for the monitor

To leave the monitor program
press: **[CTRL][BREAK]**

Additional Econet commands

This section describes a set of Econet commands which are not covered in one or both of the other Econet user guides, because their use can affect the security of the network.

Copying another station's screen *VIEW

Type: ***VIEW <station number>[RETURN]**

to copy the screen of the station number specified on to your screen.

If the screen of the station you wish to view is in a screen mode with a higher number than your own, it will change the mode of your screen to that of the other station.

If the number is lower you will get an error message, "Mode <number>". This is to prevent the change of mode overwriting your BASIC workspace. You can read the remote machine's mode after such an error: instructions are in the section *Econet error handling*.

EXAMPLE

if you *want* to *view a* screen *in mode 3 while* yours is in mode 7 you *will see the error* message "Mode 3" *on* your screen.

You can also view another user by specifying the user identifier.

Type: ***VIEW JPB[RETURN]**

to copy JPB's screen on to yours, if JPB is logged on to the file server. If he is logged on at more that one station you will copy the screen of the station at which he logged on first.

You can include *VIEW in programs. For example, if you want to stop your own prompt appearing on your screen in the middle of the one you have copied, write the following short BASIC program to prevent the prompt appearing until you press [ESCAPE].

To view station 100

```
10 *VIEW 100
20 REPEAT:UNTIL0:REM LOOP FOREVER
```

Taking over another station *REMOTE and *ROFF

This takes over the other machine completely and disables its keyboard.

Type: ***REMOTE <station number>[RETURN]**

EXAMPLE

To take over *the* user J*OE at station 200, if you* are at *station 100*

type: *REMOTE 200[RETURN]
or: *REMOTE JOE[RETURN]

If you now type *RUN at* station *100, whatever instructions you type will be* carried out at *JOE's station and he will have no control over it.*

To sever the connection

type: ***ROFF[RETURN]**

Pressing [BREAK] at the station which has been taken over, or switching it off, does not break the remote link. Pressing [BREAK] on your machine may cause "Not Listening" or "No Reply" messages on the remote machine, if it tries to communicate with you after the link is broken.

Sending short messages *NOTIFY

You can send a one-line message to another station using *NOTIFY.

Type: ***NOTIFY <station> <message>[RETURN]**
or: ***NOTIFY <user id> <message>[RETURN]**

EXAMPLE

To send a message from station 100 to *JOE at station 200*

type: *NOTIFY 200 HOW ARE YOU? [RETURN]

The message *will* go into the *keyboard buffer of JOE's machine, which will beep and print*

--100: HOW ARE YOU? –

No carriage *return* is *printed so JOE can* delete the message before *continuing*.

Protecting your station *PROT and *UNPROT

You can stop other users using *REMOTE, *VIEW and *NOTIFY on your station by typing

***PROT[RETURN]**

and remove the protection by typing

***UNPROT[RETURN]**

Any station which tries to contact yours using *NOTIFY after it has been protected will get a "Not listening" message. If a user tries to use *REMOTE or *VIEW the keyboard will be disabled until [ESCAPE] is pressed.

NOTE: if you are using file server software version 3.34 station numbers 240-254 are known as privileged stations in the Econet, and are able to by-pass this protection.

Filing system programming

This section describes the six operating system routines used for programming with the Econet filing system:

- OSFILE
- OSFIND
- OSARGS
- OSBPUT
- OSBGET
- OSGBPB.

On exit, in general:

- N, V and Z are undefined
- the interrupt state is preserved.

Interrupts may be enabled during the operation.

OSFILE

file or directory information

entry address **&FFDD**		vector address **&212**
entry	**A**	function code
	X, Y	pointer to control block
exit	**A**	*in* function *&05:* type of object found
		in function *&06:* 0 if object not found
	X	undefined
	Y	undefined
	C	undefined

control block

0	address of filename
2	reload address
6	execution address
10	start address or length
14	end address or attributes
18	

functions

&FF	loads file at specified reload address, or, if low byte of execution address parameter is not zero, at file's own reload address. Control block is updated with file's catalogue information (load address, execution address, length, attributes).
&00	saves file, and reads its catalogue information into control block.
&01	changes file's reload address, execution address and attributes to values given.
&02	changes file's reload address to value given. Only reload address need be given in control block.
&03	changes file's execution address to value given. Only execution address need be given in control block.
&04	changes file's attributes to values given. Only attributes need be given in control block.

&05 reads object's catalogue information into control block. On exit, A indicates type of object:

&00 no object found
&01 file found
&02 directory found.

&06 deletes object. If no object found, A is set to 0 on exit.

file attributes

A file's attributes are held as a four-byte item, arranged in the control block as follows:

byte	**bits**	**state**	**meaning**
14	7		undefined.
	6		undefined
	5	0	not writable by other users
		1	writable by other users
	4	0	not readable by other users
		1	readable by other users
	3	0	not locked
		1	locked
	2		undefined
	1	0	not writable by owner
		1	writable by owner
	0	0	not readable by owner
		1	readable by owner
15			days
16	0 to 3		months
	4 to 7		years since 1981
17			undefined

OSFIND **opening and closing objects**

entry address **&FFCE** vector address **&21C**

entry	**A**	function code
	X, Y	in functions &40, &80 and &C0: pointer to filename
exit	**A**	in functions &40, &80 and &C0: channel number
	X	undefined
	Y	undefined
	C	undefined

functions

&40	opens file for input only; returns channel number in A.
&80	creates and opens file for update; returns channel number in A.
&C0	opens file for update; returns channel number in A. If file does not exist, A=0.
&00	closes channel number in Y; if Y is 0, closes all open objects.

OSARGS **attributes of open object**

entry address **&FFDA** vector address **&214**

entry	**A**	function code
	X	address of first of four locations in zero page
	Y	channel number; if 0, selects special operation

exit	**A**	*in* function *&00:* 0 or, if Y=O on entry, indicates filing system type *in* function *&01:* 0 or, if Y=O on entry, undefined *in function &02:* 0 or, if Y=O on entry, indicates version number of NFS
	X	undefined
	Y	undefined
	C	undefined

functions

&00 puts value of sequential pointer into locations indicated by X, and returns 0 in A. If Y is 0 on entry, returns in A a value to indicate current filing system:

0 no current filing system
1 cassette, 1200 baud
2 cassette, 300 baud
3 ROM
4 disc
5 Econet
6 teletext/prestel.

&01 updates sequential pointer from locations indicated by X; if pointer is moved past file end, file is extended with zeros and zero is returned in A. If Y is 0 on entry, returns pointers to the command line of the last command sent to the file server in the locations indicated by X, so that decoding of a command line can take place inside a program loaded as a command. In NFS version 3.34, pointer is set to position in command line immediately after +. In version 3.40, pointer is set to first non-space position after command itself.

&02 puts file's extent into location indicated by X, and returns 0 in A. If Y is 0 on entry, returns in A a value to indicate NFS version number:

2	NFS version 3.34
1	NFS version 3.40 or greater.

OSBPUT **write byte**

entry address **&FFD4**		vector address **&218**
entry	**A**	byte to be written
	X	undefined
	Y	channel number
exit	**A**	preserved
	X	preserved
	Y	preserved
	C	undefined

function

writes byte to file at position of sequential pointer.

OSBGET **read byte**

entry address **&FFD7**		vector address **&216**
entry	**A**	undefined
	X	undefined
	Y	channel number
exit	**A**	byte read if operation successful, &FE if reading the byte after end of file
	X	preserved
	Y	preserved
	C	1 if reading the byte after end of file, 0 otherwise

function

reads byte from file at position of sequential pointer.

OSGBPB **read and write bytes**

entry address **&FFD1**		vector address **&21A**
entry	**A**	function code
	X, Y	pointer to control block
exit	**A**	undefined
	X	undefined
	Y	undefined
	C	undefined

control block

0	channel number (in *function* &08: cycle number)
1	data pointer
5	number of bytes (in function &08: number of files)
9	number of bytes (in function &08: number of files)
13	

functions

&01 writes specified number of bytes to file, from area of memory pointed to by data pointer. File pointer indicates where in file writing should start. On exit, file pointer updated to point to byte after last one written to file, and "number of bytes" field indicates how many bytes were not transferred.

&02 writes specified number of bytes to file, from area of memory pointed to by data pointer. Sequential pointer indicates where in file writing should start. On exit, "number of bytes" field indicates how many bytes were not transferred.

&03 reads specified number of bytes to area of memory pointed to by data pointer. File pointer indicates where in file reading should start. On exit, file pointer updated to point to byte after last one read from file, and "number of bytes" field indicates how many bytes were not transferred.

&04 reads specified number of bytes to area of memory pointed to by data pointer. Sequential pointer indicates where in file reading should start. On exit, "number of bytes" field indicates how many bytes were not transferred.

&05 using data pointer, reads into memory:
- length of disc title (1 byte)
- disc title (ASCII string, up to 16 bytes)
- user's option (1 byte).

&06 using data pointer, reads into memory:
- a zero byte
- length of name of currently-selected directory (1 byte)
- directory name (ASCII string, up to 10 bytes)
- &00 for owner access or &FF for public access.

&07 using data pointer, reads into memory:
- a zero byte
- length of name of currently-selected library (1 byte)
- library name (ASCII string, up to 8 bytes)
- &00 for owner access or &FF for public access.

&08 reads specified number of filenames into memory.

Returns filenames in order, each preceded by a length byte and left-justified in a field of ten characters padded with spaces.

Attempts to transfer more than 250 bytes in one operation will give a "No reply" error.

The directory pointer points to the number in directory where reading should start: 0 refers to first filename, 1 to second, and so on. If no entries are transferred, directory pointer given points outside number of entries in directory. Pointer is incremented by number of entries transferred: it should not be more than 255.

Network programming

This section describes:

- the three co-operative operations
- the seven immediate operations
- five useful OSWORD and OSBYTE calls.

TRANSMIT

Setting up the TRANSMIT control block
call: **OSWORD**

entry address **&FFF1**		vector address **&20C**
entry	**A**	&10
	X, Y	pointer to TRANSMIT control block
exit	**A**	undefined
	X	undefined
	Y	undefined
	C	undefined

control block

0	control byte: on entry, set top bit; byte will be zero if transmission fails to start
1	destination port
2	destination station
4	address of buffer start
8	address of buffer end
12	

Polling transmission
call: **OSBYTE**

entry address **&FFF4**		vector address **&20A**
entry	**A**	&32
	X	undefined
	Y	undefined

exit	**A**	undefined		
	X	status of transmission: 0 if transmission successfully completed, otherwise:		
		bit	**state**	**meaning**
		7	0	completed
			1	in progress
		6	0	successful
			1	failed
		5	0	
		0-4		error code if failed, zero if not
	Y	undefined		
	C	undefined		

error codes

&40	line jammed
&41	some part of four-way handshake lost or damaged
&42	no scout acknowledgement in four-way handshake
&43	no clock
&44	bad TRANSMIT control block

RECEIVE

Setting up the RECEIVE control block
call: **OSWORD**

entry address **&FFF1**		vector address **&20C**
entry	**A**	&11
	X, Y	pointer to RECEIVE control block
exit	**A**	undefined
	X	undefined
	Y	undefined
	C	undefined

control block

0	&00: is set to control block number on exit; will be set to 0 if unable to open block because space limit reached
1	flag: must be set to &7F on entry; when transmission is received, control byte from remote station's TRANSMIT control block is copied here
2	port
3	station
5	address of buffer start
9	address of buffer end
13	

The network software can open only a limited number of control blocks (about 16).

Polling reception
call: **OSBYTE**

entry address **&FFF4**		vector address **&20A**
entry	**A**	&33
	X	RECEIVE control block number
	Y	undefined

exit	**A**	undefined
	X	flag: if the top bit is set, a message has been received
	Y	undefined
	C	undefined

Reading the RECEIVE control block
call: **OSWORD**

entry address **&FFF1** vector address **&20C**

entry	**A**	&11
	X, Y	pointers to area into which RECEIVE control block will be copied

The routine copies the fields of the RECEIVE control block from the flag down into memory, deleting the original.

area reserved

0	RECEIVE control block number
1	*area for* RECEIVE *control block:* flag
2	port
3	station
5	address of buffer start
9	address of buffer end
13	

The transmitting station, port number and new address of the buffer end are given in the block.

Deleting a RECEIVE control block
call: **OSWORD**

entry address **&FFF4**		vector address **&20A**
entry	**A**	&34
	X	control block number
exit	**X**	control block number

BROADCAST

A special version of TRANSMIT. Sends eight bytes direct from transmit control block to every station with a RECEIVE block set for station &00, and the appropriate port number. Bytes 4 to 11 in the transmit control block contain the data to send. In software versions 3.34 and 3.40 data arrives in the “data pointer” fields of the RECEIVE control block, in later versions it arrives in the RECEIVE buffer. The BROADCAST transmit control block is as follows.

control block

0	control byte
1	destination port
2	&FF
4	&FF
8	data (8 bytes only)
12	

PEEK

Copies a block of memory from remote station into a buffer in your station.

Setting up the PEEK control block
call: **OSWORD**

entry address **&FFF1**		vector address **&20C**
entry	**A**	&10
	X, Y	pointer to control block

control block

0	&81
1	&00
2	remote station
4	address of buffer start
8	address of buffer end
12	address of data start in remote station
16	

Polling transmission
call: **OSBYTE**

entry address **&FFF4**		vector address **&20A**
entry	**A**	&32
	X	undefined
	Y	undefined
exit	**as in polling in TRANSMIT**	

POKE

Copies the contents of a buffer in your station into memory at remote station.

Setting up the POKE control block
call: **OSWORD**

entry address **&FFF1** vector address **&20C**

entry	**A**	&10
	X, Y	pointer to control block

control block

0	&82
1	&00
2	remote station
4	address of buffer start
8	address of buffer end
12	address of data start in remote station
16	

Polling transmission
call: **OSBYTE**

entry address **&FFF4** vector address **&20A**

entry	**A**	&32
	X	undefined
	Y	undefined
exit	**as in polling in TRANSMIT**	

JSR

Sends an argument block to remote station and passes control to the routine at the specified address in the remote station.

Sending the argument block
call: **OSWORD**

entry address **&FFF1**	vector address **&20C**

entry **A** &10
X, Y pointer to control block

control block

0	&83
1	&00
2	remote station
4	address of argument block start
8	address of argument block end
12	call address in remote station
16	

The routine is entered with interrupts disabled, and they should be enabled if the routine takes more than about a millisecond to complete. The remote routine can read the data passed to it using OSWORD with A set to &12. The JSR user procedure and operating system procedure bits in the protection mask of the remote station are automatically set to prevent the argument block buffer from being overwritten by another call before it is read, though this protection does not apply to calls from privileged stations. The routine should return with RTS. Maximum size of argument block: 128 bytes.

Reading the argument block
call: **OSWORD**

entry address **&FFF1**	vector address **&20C**

entry	**A**	&12
	X, Y	pointer to control block

On exit, first two bytes of control block hold number of calling station. Contents of argument block buffer follow.

The JSR user procedure, and operating system procedure bits in the protection mask are automatically set back to their previous states by this call.

Finding the number of arguments received
call: **OSWORD**

entry address **&FFF1**	vector address **&20C**

entry	**A**	&13
	X, Y	pointer to control block

control block

0	&09
1	&00; will be set to number of arguments
2	&00; will be set to argument block buffer size

User procedure call

Sends an argument block to remote station, and causes event number 8.

Setting up the user procedure call control block
call: **OSWORD**

entry address **&FFF1** vector address **&20C**

entry	**A**	&10
	X, Y	pointer to control block

control block

0	&84
1	&00
2	remote station
4	address of argument block start
8	address of argument block end
12	procedure number
14	

See *BBC User Guide* for description of events. If event number 8 is enabled, the event routine is entered with the procedure number on X, Y. Event number 8 is enabled by *FX14,8 and disabled by *FX13,8. Argument block is accessed as described for the JSR operation and notes there on protection apply.

Polling transmission
call: **OSBYTE**

entry address **&FFF4** vector address **&20A**

entry	**A**	&32
	X	undefined
	Y	undefined
exit	**as in polling in TRANSMIT**	

Machine type

Returns code to indicate machine type and NFS version of remote station. Equivalent to a PEEK of an area of Econet software in the station.

Setting up the "machine type" control block
call: **OSWORD**

entry address **&FFF1** vector address **&20C**

entry **A** &10
X, Y pointer to control block

control block

0	&88
1	&00
2	remote station
4	address of buffer start
8	address of buffer end
12	

The first four bytes in the remote machine are relevant:

byte	**value**	**meaning**
1	&01	BBC Microcomputer
	&02	Acorn Atom
	&03	Acorn System 3 or 4
	&04	Acorn System 5
2	&00	
3	&34	NFS software version x.34
	&35	NFS software version x.40
4	&03	NFS software version 3.xx

Polling transmission
call: **OSBYTE**

entry address **&FFF4**		vector address **&20A**
entry	**A**	&32
	X	undefined
	Y	undefined
exit	**as in polling in TRANSMIT**	

Halt

Halts foreground process in remote station, so that its memory can be read and written without interference from any program running.

Setting up the "halt" control block
call: **OSWORD**

entry address **&FFF1** vector address **&20C**

entry	**A**	&10
	X, Y	pointer to control block

control block

0	&86
1	&00
2	remote station
4	

Interrupts are not disabled, so operating system functions and user background processes are allowed to continue. A second processor fitted to remote station will not be halted, unless it communicates with halted input/output processor.

Polling transmission
call: **OSBYTE**

entry address **&FFF4** vector address **&20A**

entry	**A**	&32
	X	undefined
	Y	undefined
exit	**as in polling in TRANSMIT**	

Continue

Allows foreground process at remote station to continue after a “halt”.

Setting up the “continue” control block
call: **OSWORD**

entry address **&FFF1**		vector address **&20C**
entry	**A**	&10
	X, Y	pointer to control block

control block

0	&87
1	&00
2	remote station
4	

Polling transmission
call: **OSBYTE**

entry address **&FFF4**		vector address **&20A**
entry	**A**	&32
	X	undefined
	Y	undefined
exit	**as in polling in TRANSMIT**	

Protection against immediate operations

call: **OSWORD**

entry address **&FFF1**		vector address **&20C**
entry	**A**	&13
	X, Y	pointer to control block
exit	**A**	undefined
	X	undefined
	Y	undefined
	C	undefined

control block

0	&05
1	value of protection mask
2	

protection mask

bit	**state**	**meaning**
0	0	PEEK allowed
	1	PEEK not allowed
1	0	POKE allowed
	1	POKE not allowed
2	0	JSR allowed
	1	JSR not allowed
3	0	user procedure call allowed
	1	user procedure call not allowed
4	0	operating system procedure call allowed
	1	operating system procedure call not allowed
5	0	“halt” allowed
	1	“halt” not allowed

The operating system procedure call is used by the *REMOTE, *VIEW and *NOTIFY commands, and is not documented in this guide. It is not possible to protect against “continue” or “machine type” calls. Machines with NFS version 3.34 cannot protect themselves against privileged stations (those with numbers 240 to 256). Machines with versions 3.40 and above can.

Station information

call: **OSWORD**

entry address **&FFF1** vector address **&20C**

entry	**A**	&13
	X, Y	pointer to control block
exit	**A**	undefined
	X	undefined
	Y	undefined
	C	undefined

control block

0	function code
1	data
4	

function codes

code	**operation**	**number of bytes**
0	read file server number	2
1	write file server number	2
2	read printer server number	2
3	write printer server number	2
4	read protection mask	1
5	write protection mask	1
6	read user environment: main directory, currently selected directory and library in order	3
7	write user environment	3
8	read local station number	2
9	read number of arguments and size of argument block buffer size (in bytes, in order)	2
10	read error number	1

Reading and writing of information related to the file server (function codes 0, 1, 6, 7) should only be carried out with the Econet selected as the current filing system.

Code 10 reads internal error number of a composite error, and returns mode number after “Mode x” error.

Communicating with the file server

call: **OSWORD**

entry address **&FFF1** vector address **&20C**

entry	**A**	&14
	X, Y	pointer to control block
exit	**A**	undefined
	X	undefined
	Y	undefined
	C	undefined

control block

0	&00
1	size of rest of block
2	&00; will be set to reply port number
3	function code
4	&00; will be set to channel number of your main directory
5	&00; will be set to channel number of your currently selected directory
6	& 00; will bc set to channel number of your currently selected library
7	rest of file server transmit block

Locations 2 to 6 form the transmit header. Of these, fields 4, 5 and 6 hold the set of channel numbers known as the user environment.

The function code selects the operation to be performed by the file server. Twenty seven different functions are available.

After the transmit header, you give any other information the file server needs to carry out the operation you have specified

When your station receives a reply from the file server, the transmit header and the transmitted data are replaced by a receive header and, in most operations, the data that has been received. The control block will now look like this:

control block

0	&00
1	size of rest of block
2	command code
3	return code
4	rest of file server receive block

The command code is zero if the operation is complete and successful, in which case the conversation ends. If non-zero, the operation continues with further messages.

The return code is zero if no errors occurred during execution of the previous command step. If not zero, the value is an Econet error number and data returned is the error message held as an ASCII string terminated by &0D (carriage return).

The program should know the maximum amount of data to expect from the current operation step and be able to accomodate it in the OSWORD control block. The RECEIVE header and data are never longer than &80 bytes.

Note that this call will only send to the filing system command port and receive at the local station reply port, and therefore cannot be used to carry out conversations which use other ports, such as in LOAD and SAVE operations.

This OSWORD routine may not be called from within an interrupt or event routine.

Changing the user environment

To read and change your user environment, use the OSWORD "Station information" call, described above. Set the function code to &06 to read your user environment or &07 to change it.

Function codes

All the 27 function codes are listed here, but only the codes likely to be useful to programmers are described in full.

function code 0	**command line decoding**
function code 1	**SAVE**
function code 2	**LOAD**
function code 3	**examine**

File server provides information about files in named directory. You give:

- position in directory of first file to be examined
- number of files to be examined.

Value of ARG selects type of information.

1/client to file server

bytes	
1-5	Standard transmit header
6	ARG
7	entry point to directory
8	number of entries
9+	name of directory

2/file server to client bytes

bytes	
1-2	standard receive header
3	number of entries examined
4+	depends on ARG

ARG 0

4-13	file title
14-17	reload address
18-21	execution address
22	access LWR/WR (bottom 5 bits)
23	date: day
24	date: year since 1981 (4 bits), month (4 bits)
25-27	system internal name
28-30	size of file

Filename is left-justified in a field of ten characters padded with spaces. Access and data in standard format. The frame of 27 bytes is repeated for each file examined.

ARG 1

4+	character string of all information

String contains all file information, ready to be printed out. A separate string, delimited by a zero byte, is given for each file. Final delimiter is followed by a &80 byte.

ARG 2

4	10 (used by BBC Microcomputer operating system)
5-14	filename

Filename justified as in ARG 0.

ARG 3

4+	filename followed by formatted access string

Delimiters as for ARG 1.

function code 4 **read catalogue header**

function code 5 **load as command**

function code 6 **OPEN**

function code 7 **CLOSE**

function code 8 **read byte**

function code 9 **write byte**

function code 10 **read bytes**

function code 11 **write bytes**

function code 12 **read random access information**

function code 13 **write random access information**

function code 14 **read disc names**

Returns information on stated number of disc drives, starting with drive specified

1/client to file server

bytes	
1-5	standard transmit header
6	first drive number
7	number of drives

2/file server to client bytes

bytes	
1-2	standard receive header
3	number of drives found
4	drive number of first drive requested
5-20	name of disc in drive
21	drive number of second drive requested
22-37	name of disc in drive
38+	information on remaining drives in same format

Disc name is left-justified in field of sixteen characters padded with spaces.

function code 15 **read logged-on users**

Returns information on stated number of logged-on users, starting with user specified.

1/client to file server

bytes	
1-5	standard transmit header
6	first user
7	number of users

2/file server to client bytes

bytes	
1-2	standard receive header
3	number of users found
4-5	machine number where user is logged on
6-15	user identifier
16	user's privilege
17+	other entries in same format

User privilege: 0 for unprivileged, 1 for privileged.
User identifier is left-justified in field of ten characters padded with spaces.

function code 16 **read date and time**

1/client to file server

bytes	
1-5	standard transmit header

2/file server to client bytes

bytes	
1-2	standard receive header
3-4	date
5	time: hours
6	time: minutes
7	time: seconds

Date is held in same format as in function code 3, ARG 0. Bytes 5-7 are zeros if file server has no time board attached.

function code 17 **read end-of-file status**

function code 18 **read object information**

function code 19 **write file information**

function code 20 **delete object**

function code 21 **read user environment**

function code 22 **write autostart option**

function code 23 **end session**

1/client to file server

bytes	
1-5	standard transmit header

2/file server to client bytes

bytes	
1-2	standard receive header

function code 24 **read user information**

1/client to file server

bytes	
1-5	standard transmit header
6+	user name

2/file server to client bytes

bytes	
1-2	standard receive header
3	date
4-5	machine number at which user is logged on

User privilege: 0 for unprivileged, 1 for privileged.

function code 25 read file server version number

1/client to file server

bytes	
1-5	standard transmit header

2/file server to client bytes

bytes	
1-2	standard receive header
3+	version number

Version number is returned as a character string, terminated by carriage return.

function code 26 read file server free space

1/client to file server

bytes	
1-5	standard transmit header
6+	disc name

2/file server to client bytes

bytes	
1-2	standard receive header
3-5	number of free blocks on disc

Sending text messages

call: **OSWORD**

entry address **&FFF1**		vector address **&20C**
entry	**A**	&14
	X, Y	pointer to control block
exit	**A**	undefined
	X	undefined
	Y	undefined
	C	undefined

control block

0	&01
1	destination station
3	message string

This call sends messages in the same way as *NOTIFY. The message string should be ended with &00 (ASCII null) or &0D (ASCII carriage return). Maximum length of control block:

- 250 bytes normally
- 128 bytes when call is made from second processor.

Do not call this routine from within an interrupt or event routine.

Causing a remote error

call: **OSWORD**

entry address **&FFF1** vector address **&20C**

entry	**A**	&14
	X, Y	pointer to control block
exit	**A**	undefined
	X	undefined
	Y	undefined
	C	undefined

control block

0	&02
1	destination station
3	

This call causes a fatal error in a remote machine, to terminate execution of a program and re-enter the current language. It can be used to make sure a text message will appear on the display. BASIC I does not recognise the fatal error: it cannot be trapped by an ON ERROR statement. BASIC II performs correctly. Do not call this routine from within an interrupt or event routine.

Severing a remote connection

call: **OSWORD**

entry address **&FFF4** vector address **&20A**

entry	**A**	&35

This routine severs a connection set up by *REMOTE: it is equivalent to *ROF'F.

Listing of example TRANSMIT and RECEIVE programs

Below are full listings of the example TRANSMIT and RECEIVE programs discussed in the section *Programming with the network.*

TRANSMIT program

```
10 REM Program to produce sounds
20 REM on a remote machine
30 REM
40 REM --------------------
50 REM
60 OSWORD=&FFF1:OSBYTE=&FFF4
70 max_tx_length%=20:REM  maximum size of transmitted packet
80 txport%=100:REM transmit port
90 DIM cblock% 13,txbuffer% max_tx_length%
100 REM Read in the station number
110 INPUT "Remote station: "S%
120 PRINT "Now press keys..."
130 REPEAT
140 key%=GET:REM Read a key
150 PRINT CHR$key%;
160 ?txbuffer%=key%:REM Put the key in the transmit buffer
170 REM and transmit a buffer of length 1 to station S%
180 tx_result%=FNTRANSMIT(S%,txport%,1)
190 UNTIL tx_result%<>0:REM Loop until a transmit fails
200 REM Then check error number
210 ON tx_result%-&3F GOTO 220,230,240,250,260
220 PRINT"Line jammed":END
230 PRINT"Net error":END
240 PRINT"Not Listening":END
250 PRINT"No clock":END
260 PRINT"Bad TRANSMIT control block":END
270 DEF FNTRANSMIT(station%,port%,length%)
280 LOCAL X%,Y%,A%,tries%,delay%,U%,txresult%,nonfatal%
290 REM First set up the control block
300 REM
310 cblock%?1=port%:REM port number
320 cblock%!2=station%:REM station number (2 bytes)
330 cblock%!4=txbuffer%:REM pointer to data buffer
340 cblock%!8=txbuffer%+length%:REM Pointer to data buffer end
350 tries%=10:delay%=50
360 REPEAT
370 REM
380 REM Now attempt to start the transmission
```

```
390 REM First, set the registers A, X and Y
400 REM (X and Y point to the control block)
410 X%=cblock%:Y%=cblock% DIV 256:A%=&10
420 REPEAT
430 ?cblock%=&80:REM Set the control byte
440 CALL OSWORD
450 UNTIL ?cblock%<>0 :REM Repeat until the transmit starts
460 REM
470 REM Now poll the TRANSMIT block until completion
480 REM
490 REPEAT
500 A%=&32:U%=USR OSBYTE
510 UNTIL (U% AND &8000)=0:REM Check top bit of the X register
520 REM
530 REM Now check if an error, and if a re-try is required
540 REM
550 txresult%=(U% AND &FF00) DIV 256:REM Mask off the X register
555 nonfatal%=txresult%=&41 OR txresult%=&42
560 IF nonfatal% THEN PROCdelay(delay%):tries%=tries%-1
570 UNTIL tries%=0 OR NOT nonfatal%
580 REM Continue to loop until either a fatal error
590 REM or a successful transmission,
600 REM or all re-tries have failed
610 REM
620 =txresult%:REM return result
630 REM
640 REM --------------------
650 REM
660 DEF PROCdelay(n%)
670 REM Wait for n% centiseconds
680 LOCAL limit%
690 limit%=TIME +n%
700 REPEAT
710 UNTIL TIME >=limit%
720 ENDPROC
```

RECEIVE program

```
10 REM Receive a key pressed by a remote station
20 REM
30 OSWORD=&FFF1:OSBYTE=&FFF4
40 max_rx_length%=20
50 DIM cblock% 13,rxbuffer% max_rx_length%
55 port%=100
60 REM
70 REM --------------------
80 REM
85 INPUT "From station: "S%
90 REPEAT
100 PROCRECEIVE(S%,port%):REM Wait for a reception from S%
110 key%=?rxbuffer%:REM Read the data sent
120 PROCsound(key%)
130 UNTIL  FALSE  :REM Repeat forever
```

```
140 REM
150 REM -------------
160 REM
170 DEF PROCRECEIVE(station%,port%)
180 REM Set up a RECEIVE block and wait for some
190 REM data to arrive from <station%> on <port%>
200 REM
210 REM First, set up the control block
215 ?cblock%=0
220 cblock%?1=&7F:REM Receive flag
230 cblock%?2=port%
240 cblock%!3=station%
250 cblock%!5=rxbuffer%:REM Pointer to buffer
260 cblock%!9=rxbuffer%+max_rx_length%:REM Pointer to buffer end
270 REM
280 REM Now call OSWORD to open the RECEIVE block
290 REM
300 X%=cblock%:Y%=cblock% DIV 256:A%=&11
310 CALL OSWORD
320 rxcb_number%=?cblock%:REM Read RECEIVE control block number
330 REM
340 REM Now wait for a reception
350 REM
360 A%=&33:X%=rxcb_number%
370 REPEAT:U%=USR OSBYTE
380 UNTIL (U% AND &8000)<>0
390 REM
400 REM A reception has happened, so read the control block back
410 REM
420 X%=cblock%:Y%=cblock% DIV 256:A%=&11
430 ?cblock%=rxcb_number%:CALL OSWORD
440 bytes_received%=cblock%!9-cblock%!5:REM Find out bytes received
450 ENDPROC:REM Reception now completed, and data is in the buffer
470 REM
480 REM -------------------
490 REM
500 DEF PROCsound(key%)
510 LOCAL keys$,pitch%
520 ENVELOPE 1,5,1,-1,1,1,1,1,126,-3,-3,-3,126,0
530 keys$="Q2W3ER5T6Y7UI9O0P@^[\_"
540 pitch%=INSTR(keys$,CHR$key%)*4
550 SOUND &11,1,pitch%,-1
560 ENDPROC
```

Econet error handling

The BBC Microcomputer can only cope with error numbers from the Econet file server in the range &A8 to &C0 (decimal 168 to 192), but the file server can generate many more errors than this range allows. To overcome this problem, &A8 is used as a composite error number, so that it covers every error with a number less than &AO.

To find the true number of an &A8 error,
call: **OSWORD**

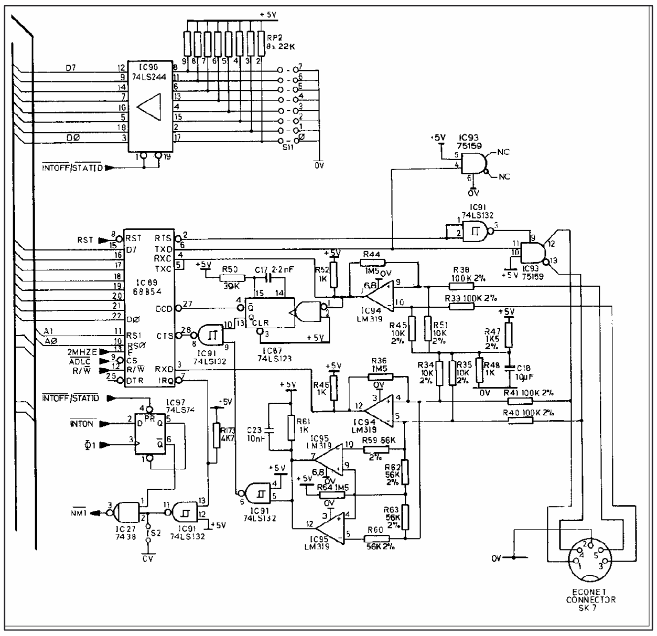
entry address **&FFF1**		vector address **&20C**
entry	**A**	&13
	X, Y	pointer to control block
exit	**A**	undefined
	X	undefined
	Y	undefined
	C	undefined

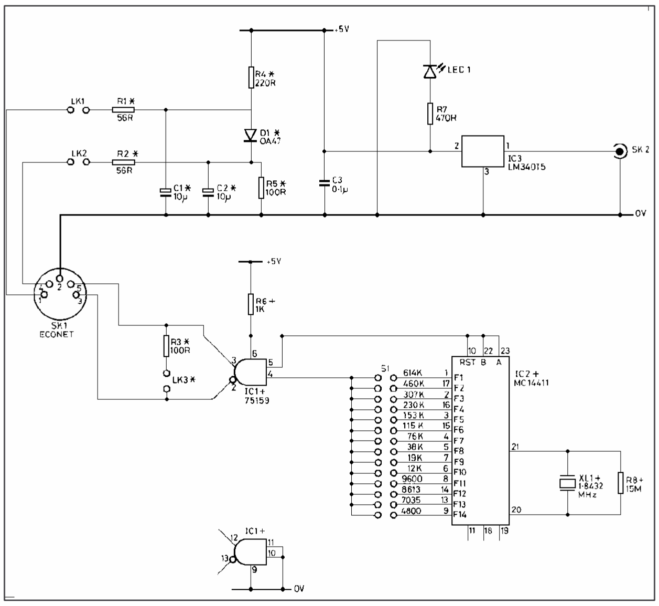
control block

0	&0A
1	&00; will be set to true error number
3	

Circuit diagrams

Econet interface on the BBC Microcomputer





Components marked * are fitted on terminator boxes only.
Components marked + are fitted on clock boxes only.

Index