7. An Application Note for Econet Programming on the BBC Microcomputer

Topics covered:

The NFS ROM
OSARGS
Printing
File handling
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

7.1 The NFS ROM

Two common versions of the NFS ROM exist. The one you will probably have is NFS 3.34 in EPROM: if you are lucky you may have a DNFS (in ROM), and if you know the right person, an NFS 3.6 EPROM which is a DNFS ROM 'cut in half'. Acorn is also developing an 'Advanced NFS', (ANFS) but it is not known when this is to be available. The main feature of ANFS is that is does its own 'local buffering' - see *PUTGET.

To find out which version you have, type ***HELP.** It is possible to read the NFS type using OSARGS (see below), which is useful for version-independent programming.

Whenever a reference is made to NFS 3.6, this also includes the DNFS ROM, as the Econet code is identical in both.

7.2 OSARGS

See EAUG p. 40 for preference, also ESUG p.36 and AUG p.337.

Documentation relating to 'Return command line address' call, which is used to access parameters for * commands, eg

***PS DAISY**

where PS is a loaded program being run, requiring access to the string **DAISY.**

OSARGS call address &FFDA with A=0, Y=0, X pointing to 4 locations in zero-page.

NFS 3.6
The address returned is the address of the first non-space character after the filename that is being RUN. This is the 'correct' value and the one returned when using a DFS.

NFS 3.34
The address returned is incorrect but reasonably well defined: it points to the name of the file being run, execpt in the case of */ <filename> where the address points to the first character after the slash: leading spaces are not stripped.

Do not assume that registers are preserved on exit from this call, even though the DFS manual says X and Y are preserved. In fact X is INC'd on exit from this particular brand of OSARGS call on the NFS.

Handling of this call should proceed as follows:

a) Find out NFS version number by using OSARGS with Y=0, A=2.
   NFS 3.34 returns A=2, NFS 3.6 returns A=1, ANFS returns A=0 (?).
   If NFS 3.34 then continue....
b) Skip leading spaces.(We are now pointing at the filename being RUN)
c) Skip non-space characters
d) Skip spaces (now pointing at right item)

7.3 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, this also being dependant on whether printing was started with a CTRL-B typed in at the keyboard or using a VDU 2. (Don't ask me why but it is believed that VDU 2 is safer).

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 loggon-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 119 of the AUG or page 422 of the UG.

7.3.1 PS Status Enquiry call

It is possible to find out the state of a print server on the network, and also to find out what printers are available in general. The utilities ***PS** and ***POLLPS** (of which only the former is available currently) can be used, and I have included the low-level interface for completness:

Client to PS
 Port: &9E
 6 bytes of data
 2-byte Function code (=1)

PS to Client
 Port: &9F
 1 byte status:
 0 - Ready
 1 - Busy with station mm.nn
 2 - Offline
 2-byte station numbers of data

The data sent to the PS is in the form of an ASCII string padded with spaces describing the name of the printer. 'PRINT ' and 'AUTO ' are two defined names: depending on how the PS is set up others may be used.

As an important corollary, sending the Status Enquiry call as a 'Broadcast' message will cause all PS's that recognise the string to reply. This is what *PS uses.

7.4 File handling

As has been drummed into you by now, file access using BGET#, BPUT# (and therefore INPUT#, PRINT#) is S L O W. The remedy is to use OSGBPB (often pronounced 'Ozheebeegeebee'). One of the problems about using this call is the lack of documentation, and also the fact that the cassette filing system (CFS) does not support OSGBPB, but only OSBGET and OSBPUT: this means that a lot of commercial software packages, wishing to be compatible with CFS, use BGET and BPUT. The only solution here is to use PUTGET (q.v.).

For those interested in writing their own file handlers, I have prepared a useful list of references and routines.

OSGBPB documentation can be found in:
DFS manual (The yellow 'n' green book) p. 73
ESUG p. 37
EAUG p. 43
AUG p. 339

The most important points to remember are these:

OSGBPB call address is &FFD1, and on entry:
YX points to a 13-byte control block (even if you are not using the given pointer)
A is a reason code:
1,2 Put bytes (1 uses the given pointer)
3,4 Get bytes (3 uses the given pointer)

On exit:
All execpt the first byte of the control block are updated.

DO NOT PUT THE CONTROL BLOCK IN PAGE ZERO. There is a bug in NFS 3.34 such that OSGBPB does not work (at all) if you do this.

From BASIC it is convenient to have a procedure such as:

```
DEFPROCgbpb(A%,handle%,buffer%,length%,offset%)
 LOCAL X%,Y%
 X%=gb%
 Y%=gb% DIV 256
 ?X%=handle%
 X%!1=buffer%
 X%!5=length%
 X%!9=offset%
 CALL &FFD1
ENDPROC
```

with a statement of the form:

```
 DIM gb% 12
```

(note gb%?0 and gb%?12 are both available so this has allowed for 13 bytes of memory)

somewhere near the beginning of the program.

As an example, the call PROCgbpb(4,F%,&3000,100,2000) would load 100 bytes from offset 2000 in the file whose handle is F% to location &3000. Normally you would have a DIM BUFF nnnn statement in your program, and use BUFF instead of the

constant &3000.

7.4.1 OPENIN

The other most important aspect of file handling on the network is the difference between OPENIN and OPENUP, and the fact that BASIC-I's OPENIN is in fact interpreted as an OPENUP. If you have BASIC-I you will have to write a special function for OPENIN, viz:

```
DEFFNopenin($ob%)
 LOCAL A%,X%,Y%
 A%=&40      (OPENIN type)
 X%=ob%
 Y%=ob% DIV 256
=(USR&FFCE) AND 255
```

where, like gb%, ob% has been DIMensioned and is an 'OPENIN buffer', and should be of the order of 80 bytes to enable it to accomodate a reasonably long filename (see below).

(See AUG p.342, ESUG p. 34, EAUG p. 40. IGNORE the UG.)

Thus instead of F%=OPENIN("file") do F%=FNopenin("file"). On BASIC-II this is not necessary.

7.4.2 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 $.JOHN.BBCprogs.....2ndrev.OLD.developmnt.Addition) of almost infinite length. A compromise length of 80 characters should be used as a minimum. Even an a DFS file name can be up to 12 characters long (:2.R.LONGEST), requiring 13 bytes of storage including the CHR$(13) on the end. Programs which disallow filenames over 7 characters long should not be considered to be programs at all.

7.5 Econet workspace

When writing * commands 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..F02 OK
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.

7.5.1 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.

7.6 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
```

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 - *SHOW would be an example: it is a program which sends all screen output down the network, to anyone who is *WATCH-ing.

In addition, overwriting filing-system workspace can often cause the micro to crash on pressing BREAK. The NFS has been known not to recover even after a Ctrl-BREAK - a power-off was necessary!

7.7 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.

In order for NETMON to be of much use one has to be familiar with the Econet line protocols: a introduction is given below, and a comprehensive and detailed description is given in section 7.8. There is also some NETMON-specific debugging output.

***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. Ignore the '1E' - even I don't know why it is printed. 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.

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 one 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 only up to 200Kbaud reliably). Unless loss of data bytes 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.

7.8 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**)