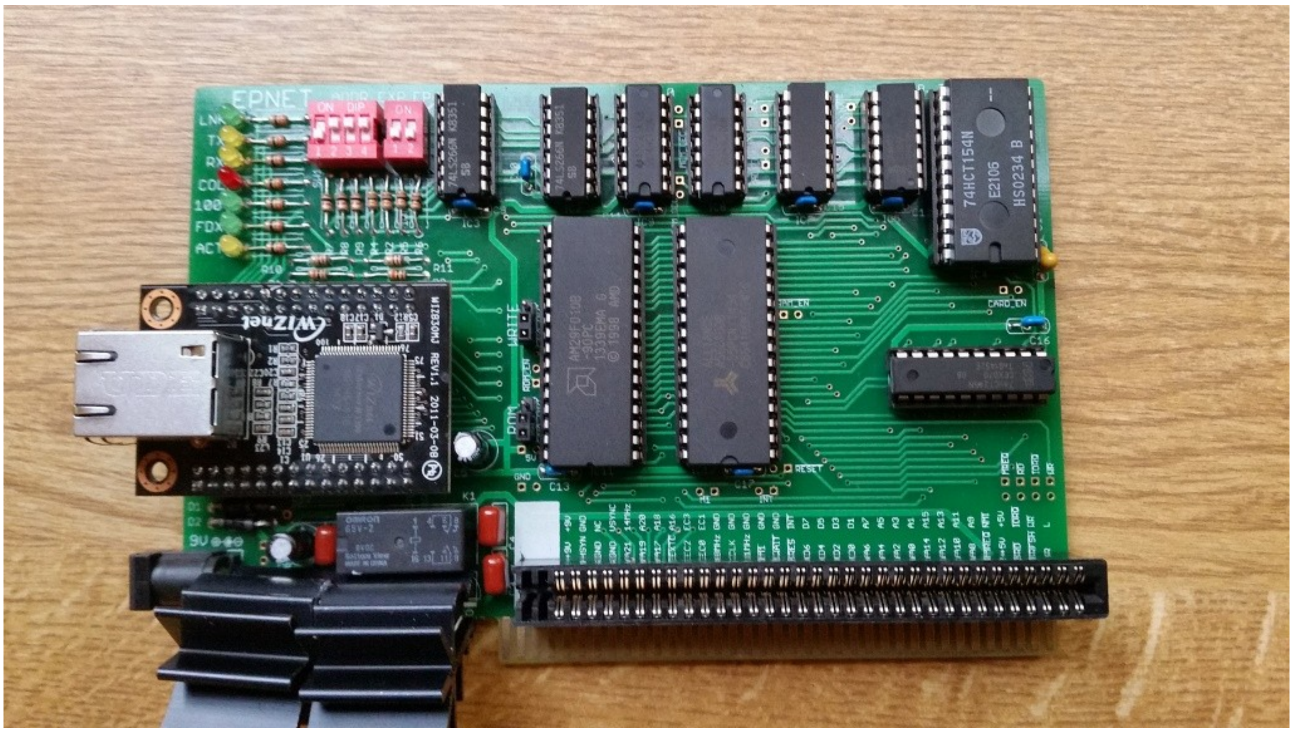
EPNET

User Manual

EPNET is an add-on card for the Enterprise computer that allows the Enterprise to connect to an Ethernet network.

It provides an extra 64k general-purpose RAM expansion for the Enterprise, but “takes back” 16k of that for its own use. For use with an Enterprise 64, the expansion RAM can be configured to turn the Enterprise 64 into an Enterprise 128 (ie. the expansion RAM appears in the right place in the Enterprise’s memory map).

Software is supplied in the 64k on-board FLASH ROM and provides a series of “: commands” to control network access, and EXOS devices which allow files to be read and written over the network. Most programs written to use EXOS should work with the network without change, most notably of course IS-BASIC, and IS-BASIC programs can be loaded and saved to a server on the network.



Licence

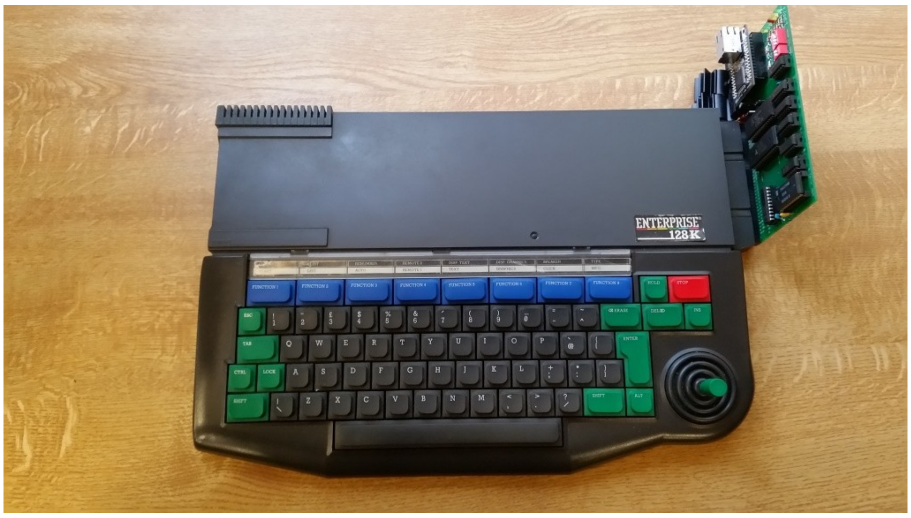
All the hardware design files and software source code files for the EPNET project are similarly available under the GNU General Public Licence, and can be freely downloaded from **www.github.com/BruceTanner/EPNET**

If you do use or modify any of the files in the EPNET project, either for their original purpose or for something new, I'd love to hear about it! I can be contacted by email at: **brucetanner@btopenworld.com**

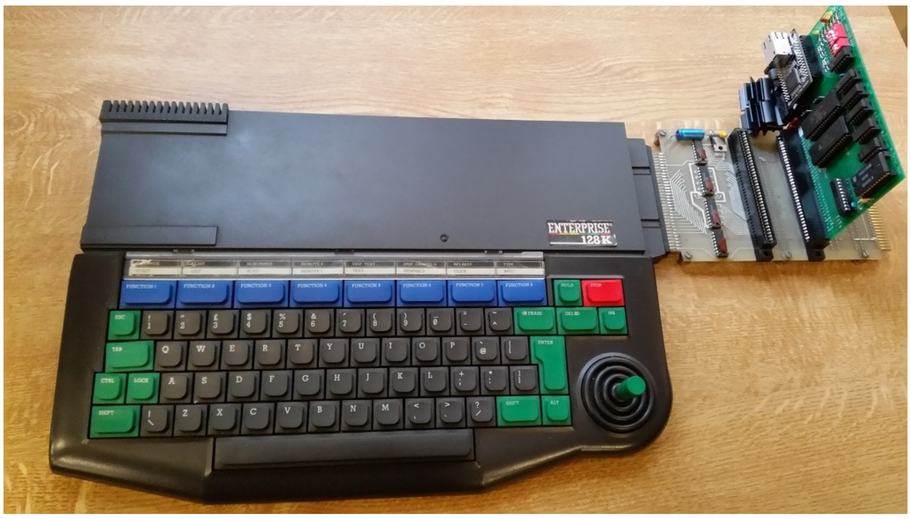
Table of Contents

Chapter 1. Quick Hardware Installation

The EPNET hardware is unusual in that it plugs into the Enterprise's right-hand side expansion port, but is used vertically, unlike most other Enterprise hardware:

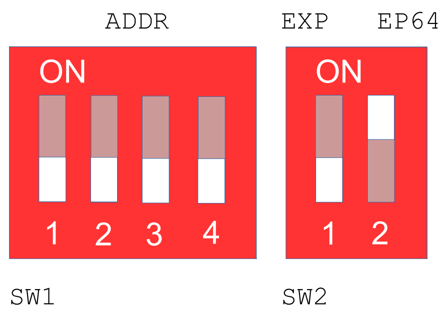


Alternatively it can be plugged into a slot on an expansion bus board:



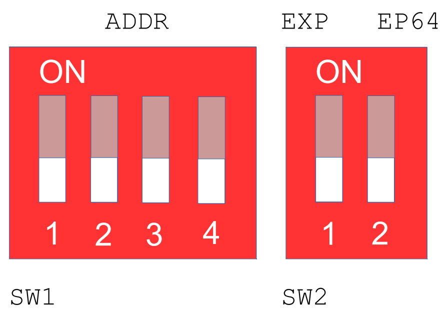
EPNET is highly configurable regarding where its I/O ports, RAM and ROM are in the Enterprise's memory map, but most applications will use one of the following three switch settings (see Appendix I on page 11 for full technical details).

1. Enterprise 64 Unexpanded



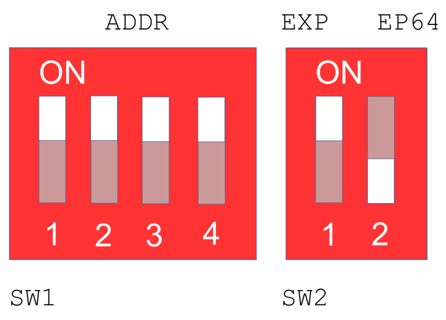
In this configuration EPNET’s expansion RAM is adjacent to the Enterprise’s on-board 64k RAM in the memory map, and EPNET’s ROM will be found by an unmodified EXOS ROM. It’s I/O ports will be at 70-77 (hex). EP64 ON, all others OFF.

2. Enterprise 128 Unexpanded



In this configuration EPNET’s expansion RAM is adjacent to the Enterprise’s on-board 128k RAM in the memory map, and EPNET’s ROM will be found by an unmodified EXOS ROM. It’s I/O ports will be at 70-77 (hex). All switches OFF.

3. Enterprise with expansion (memory, SD card reader etc)



In this configuration EPNET’s expansion RAM and ROM are tucked away a normally unused part of the memory map so they are unlikely to clash with anything else. An upgraded EXOS ROM (to 2.1 or later) will be required. It’s I/O ports will be at 00-07 (hex). EP64 OFF, all other switches ON.

Many more configurations than the three shown are possible. EPNET also uses I/O locations which must not clash with any other expansion hardware - see Appendix I on page 11 for full technical details.

Getting Started

Plug one end of a standard ethernet RJ45 “patch cable” into EPNET’s rear socket and the other end into a standard Ethernet hub or switch. When the Enterprise is powered on the green “Link” LED should then be lit continuously. If it is flashing there is something wrong with the cable or hub and EPNET will not work until the problem is fixed.

Chapter 2. Software

EPNET's software is supplied in FLASH "ROM" so no other software is necessary to start using EPNET.

From the IS-BASIC command line:

:help

should include EPNET in the list of installed software, and:

:help net

will print out a list of EPNET's available commands.

:help net *command*

will print out help for the specified :NET *command*.

:net diag

will start the EPNET hardware in diagnostic mode, which will check everything is working ok.

In normal EPNET use:

:net start

will do the same thing but without all the diagnostic messages. But in fact it is not usually necessary to "manually" start the network in this way because EPNET starts automatically when the network is first accessed, but it can be useful for testing etc.

If you know the IP address of another computer on the network, eg. a router or server:

:ping 192.168.1.100

(substitute your own IP address) will send some test packets to that computer and make sure the Enterprise can receive the response. (If you hear a "ping" sound you will have to use :net ping instead – there is a clash of : commands with another system extension.) Note the Enterprise cannot receive its own network packets and cannot "loopback"; therefore you cannot "ping yourself".

It is now possible to load and save files over network from the Enterprise. A number of protocols are available.

FTP

If you want to use the network to load and save Enterprise programs to a directory on a desktop computer, FTP is probably the easiest and most flexible way.

You will need some FTP server software to run on a Windows- or Linux-based desktop computer. This is freely available eg. Mozilla FTP Server. With this you can specify a directory to share on the network over FTP without compromising the security of the rest of the desktop computer, and create user accounts with passwords (although these are not very secure passwords by modern standards as they are sent over the network unencrypted plain text). If your desktop computer OS has a firewall built in, you will probably have to change its configuration slightly to allow incoming FTP connections.

Once the server has been set up with a user account, password and shared directory, on the Enterprise:

:ftp login 192.168.1.100

(substitute the IP address of your FTP server) will log you in to your FTP session. You will be prompted for a user name and password, with defaults of Anonymous for the user name and user@localhost for the password (so for maximum convenience set the account on the server to these. Many FTP servers do not mind what the password is, but expect something that looks like an email address).

Once logged in you can save a BASIC program with

```
save “ftp:myfile.bas”
```

and load it again with

```
load “ftp:myfile.bas”
```

There are a number of other :FTP commands that can be used. These are designed to be as similar as possible to EXDOS to make using the network as easy as using a local disk:

:ftp dir

prints a directory listing of your directory on the FTP server, and after the above example should show the file *myfile.bas*.

:ftp del *myfile.bas*

deletes the file *myfile.bas*.

:ftp ren *myfile.bas newfile.bas*

renames *myfile.bas* to *newfile.bas*.

:ftp md *mydir*

creates a new directory called *mydir*.

:ftp cd *mydir*

changes the current directory to the new one.

:ftp cd

prints the current directory

:ftp cd ..

changes the current directory to the parent of the current directory.

:ftp rd *mydir*

removes the directory called *mydir*.

:ftp logout

logs out you out from the FTP server. You can log in again with **:ftp login** as above.

You are not restricted to BASIC’s load and save – any application that opens EXOS channels should work over the network. And you can open a channel to a network file yourself, eg:

```
open #1:”ftp:myfile.txt” access output
list #1
close #1
```

Will list the current BASIC program to a file called *myfile.txt*. (`access output` is required if you are creating and writing to a new network file rather than reading an existing file.)

HTTP

The HTTP protocol allows files to be read from a HTTP (ie. web) server on the network. For example:

```
load “http:192.168.1.100/prog.bas”
```

will look for *prog.bas* on the server at IP address 192.168.1.100.

Unlike with FTP, it is not necessary to login to an HTTP server.

Again it is fairly easy to set up an HTTP server on a desktop computer, and free ones are available running under Windows or Linux etc.

As the HTTP protocol was designed to share files for reading only (ie. Web pages) it is not currently possible to save to an HTTP server or to print a directory listings etc.

TCP

tbd

Other :NET commands

:net status

will print EPNET’s unique MAC address, IP address, subnet mask, gateway and information about open network sockets.

The MAC address is built into the FLASH ROM and there is no way of changing it (other than reprogramming the FLASH).

The IP address, subnet mask and gateway are normally assigned to EPNET automatically using the DHCP protocol when it first starts on the network, but can be set manually – see **:net set**.

:net time

Sets the Enterprise’s time and date from the network using the NTP protocol.

:net trace

EPNET has a trace mode for debugging interoperability problems with other network devices and servers. For example, when in trace mode each command sent and response received from an FTP or HTTP server will be printed out as it occurs.

There is also a “raw” trace mode in which the first 16 bytes sent or received are also printed out.

`:net trace`	turns on trace mode
`:net trace raw`	turns on raw mode too
`:net trace off`	turns off trace mode

See Appendix II on page 14 for a description of typical trace output.

:net set

Sets a named variable to the specified value. There are many special named variables which can control how EPNET behaves. They are saved in FLASH memory on EPNET so will persist over power-off.

For example, EPNET normally gets its IP address using the DHCP protocol. Most home networks will have a router which will act as a DHCP server and control IP addresses used on the network. However it is possible to disable this on EPNET and specify the IP address manually:

```
:net set dhcp=n
```

will stop EPNET using DHCP (any word beginning with 'n' will work). It is then necessary to specify the IP address and other IP parameters:

`:net set ip=192.168.1.100`	(or whatever IP address you wish to use)
`:net set subnet=255.255.255.0`	(or whatever subnet mask you wish to use)
`:net set gateway=192.168.1.255`	(or whatever gateway IP you wish to use)

You can start using DHCP again with:

```
:net set dhcp=
:net start
```

which will delete the DHCP variable and restart EPNET. You can print all the variables with

```
:net set
```

eg:

```
:net set
ip=192.168.1.100
subnet=255.255.255.0
gateway=192.168.1.255
```

EPNET with EXDOS

EPNET is designed to be compatible EXDOS. From the command line interpreters of both EXDOS and ISDOS, EPNET commands can be entered (without the preceding :).

However it is not currently possible to use DIR etc with a network resource – use FTP DIR etc instead.

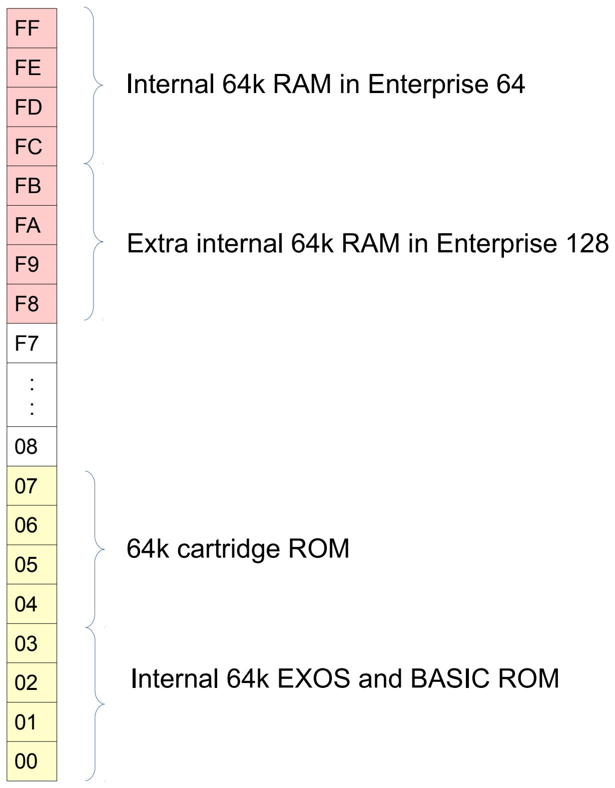
It is possible to copy a file to or from a network resource with the EXDOS copy command, eg COPY A:\myfile FTP:myfile /b or COPY FTP:myfile /b A:\myfile, but note the following points:

a) it is necessary to specify the filename for both the FTP: part and drive letter part. If you specify just FTP: a default filename will be used
b) it may be necessary to use /b after the FTP: filename. This is because EXDOS will see FTP: is a device, and will default to ASCII mode. In ASCII mode files are read up to the first ^Z character when reading, and add a ^Z to the end when writing.

Appendix I. Switch Settings

Switches SW1 and SW2 control where EPNET's RAM, FLASH ROM, and I/O ports appear in the Enterprise's memory map.

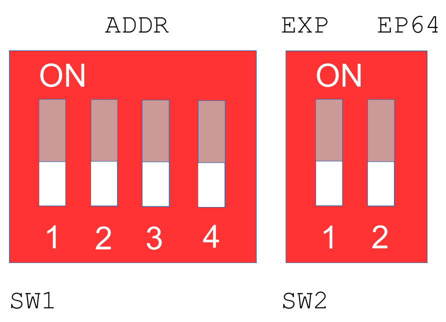
The Enterprise's memory map has a 4Mb capacity, consisting of 256 16Kb segments which can be numbered 00 to FF (hex). Each 16Kb segment can be RAM or ROM. In a standard unmodified Enterprise the memory map is:


FF
FE
FD
FC
FB
FA
F9
F8
F7
:
08
07
06
05
04
03
02
01
00
Internal 64k RAM in Enterprise 64
Extra internal 64k RAM in Enterprise 128
64k cartridge ROM
Internal 64k EXOS and BASIC ROM

Another way of illustrating this, which we shall use later, is:

	n0	n1	n2	n3	n4	n5	n6	n7	n8	n9	nA	nB	nC	nD	nE	nF
Fx	F0	F1	F2	F3	F4	F5	F6	F7	EP128 RAM 64K				EP64 RAM 64K			
Ex	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
Dx	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
Cx	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
:																
:																
3x	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
2x	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
1x	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
0x	EXOS ROM 64K				Cartridge ROM 64K				08	09	0A	0B	0C	0D	0E	0F

EPNET's ROM and RAM must be put into unused segments. This is controlled by SW1 and SW2 on EPNET (the red switch blocks at the top left of the EPNET board). The I/O locations used by EPNET must also be put into an unused area. On an unexpanded Enterprise this can be anywhere between 00 and 7F (hex).



UP=switch is ON

DOWN=switch is OFF

SW1 controls the top “nibble” of the segment number of EPNET’s ROM and RAM. SW2-EXP controls whether the ROM/RAM is at n0-n7 or n8-nF where n is the top nibble set by SW1. SW2-EP64 controls a special mode for an unexpanded Enterprise 64 which puts EPNET's RAM adjacent to the built in 64k RAM in the memory map, which some badly-written programs require.

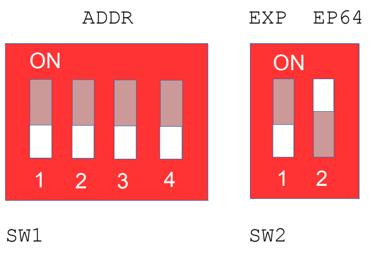
EPNET also uses 16 I/O locations. The first of these 16 is always the ROM segment number divided by 2. So:

SW1				SW2					I/O	Recommended for
1	2	3	4	EP64	EXP OFF		EXP ON**			
					RAM	ROM	RAM	ROM		
□	□	□	□	■	F0-F3	F8-FB	Conflict with internal video RAM		7x	Unexpanded Enterprise 64
□	□	□	□	□	F0-F3	F4-F7			7x	Unexpanded Enterprise 128
□	□	□	■	□	E0-E3*	E4-E7	E8-EB	EC-EF	7x	
□	□	■	□	□	D0-D3	D4-D7	D8-DB	DC-DF	6x	
□	□	■	■	□	C0-C3*	C4-C7	C8-CB	CC-CF	6x	
□	■	□	□	□	B0-B3	B4-B7	B8-BB	BC-BF	5x	
□	■	□	■	□	A0-A3*	A4-A7	A8-AB	AC-AF	5x	
□	■	■	□	□	90-93	94-97	98-9B	9C-9F	4x	
□	■	■	■	□	80-83*	84-87	88=8B	8C-8F	4x	
■	□	□	□	□	70-73	74-7F	78-7B	7C-7F	3x	
■	□	□	■	□	60-63*	64-67	68-6B	6C-6F	3x	
■	□	■	□	□	50-53	54-57	58-5B	5C-5F	2x	
■	□	■	■	□	40-43*	44-4F	48-4B	4C-4F	2x	
■	■	□	□	□	30-33	34-3F	38-3B	3C-3F	1x	
■	■	□	■	□	20-23*	24-2F	28-2B	2C-2F	1x	
■	■	■	□	□	10-13	14-17	18-1B	1C-1F	0x	
■	■	■	■	□	Conflict with internal ROM		08-0B	0C-0F	0x	Expanded Enterprise with upgraded EXOS ROM

□=off ■=on
*Standard EXOS and Expansion Bus slot addresses
**Requires upgraded internal EXOS ROM (2.1 or later)

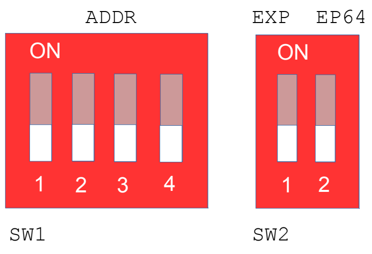
Therefore the memory map of the 3 recommended configurations mentioned in Chapter 1 on page 4 and in the table above is:

Unexpanded Enterprise 64



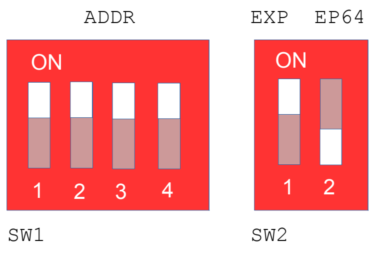
	n0	n1	n2	n3	n4	n5	n6	n7	n8	n9	nA	nB	nC	nD	nE	nF
Fx	**EPNET** ROM 64K				F4	F5	F6	F7	**EPNET** RAM 64K				EP64 RAM 64K			
Ex	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
Dx	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
Cx	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
:																
:																

Unexpanded Enterprise 128



	n0	n1	n2	n3	n4	n5	n6	n7	n8	n9	nA	nB	nC	nD	nE	nF
Fx	**EPNET** ROM 64K				**EPNET** RAM 64K				EP128 RAM 64K				EP64 RAM 64K			
Ex	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
Dx	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
Cx	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
:																
:																

Expanded Enterprise with upgraded EXOS ROM



:																
:																
3x	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
2x	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
1x	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
0x	EXOS ROM 64K				Cartridge ROM 64K				**EPNET** ROM 64K				**EPNET** RAM 64K			

Appendix II. Trace Output

When diagnostic tracing is enabled with **:net trace**, EPNET outputs diagnostic messages as it performs various interactions with the network and servers on the network. This can be useful for diagnosing interoperability problems between EPNET and other computers or devices on the network.

The higher-level protocols in EPNET work using “sockets”, a lower-level interface to the network. For example **:ftp login** will open a socket, connect to the remote FTP server using that socket, read and/or write to that socket, and then close the socket. There are 8 sockets available for the protocols to use. Diagnostic trace output includes both the high level
Protocol and the low level socket information.

Raw mode tracing is also available. In addition to the above, the actual bytes being send and received over the network are displayed.

Although trace mode will work on a 40 column screen, an 80 column screen is recommended as there can be a lot of output.

Here are some sample outputs from trace mode, starting with perhaps the simplest case: **:PING**.

Simple Example: :ping

The normal output for the :PING command is:

```
:ping 192.168.1.100
Pinging 192.168.1.100…<20mS
Pinging 192.168.1.100…<20mS
Pinging 192.168.1.100…<20mS
Pinging 192.168.1.100…<20mS
Pinging 192.168.1.100…<20mS
```

(EPNET prints “<20mS” because 20mS is the smallest time interval EXOS supports.)

The same but with trace mode on:

```
:net trace
ok
:ping 192.168.1.100
S0:Open IP port 1 by PING...OK
Pinging 192.168.1.100...
S0:IPRAW tx to 192.168.1.100 8 bytes...OK
S0:IPRAW rx from 192.168.1.100 8 bytes 460mS
Pinging 192.168.1.100...
S0:IPRAW tx to 192.168.1.100 8 bytes...OK
S0:IPRAW rx from 192.168.1.100 8 bytes 460mS
Pinging 192.168.1.100...
S0:IPRAW tx to 192.168.1.100 8 bytes...OK
S0:IPRAW rx from 192.168.1.100 8 bytes 460mS
Pinging 192.168.1.100...
S0:IPRAW tx to 192.168.1.100 8 bytes...OK
S0:IPRAW rx from 192.168.1.100 8 bytes 460mS
Pinging 192.168.1.100...
S0:IPRAW tx to 192.168.1.100 8 bytes...OK
S0:IPRAW rx from 192.168.1.100 8 bytes 460mS
S0:Close…OK
ok
```

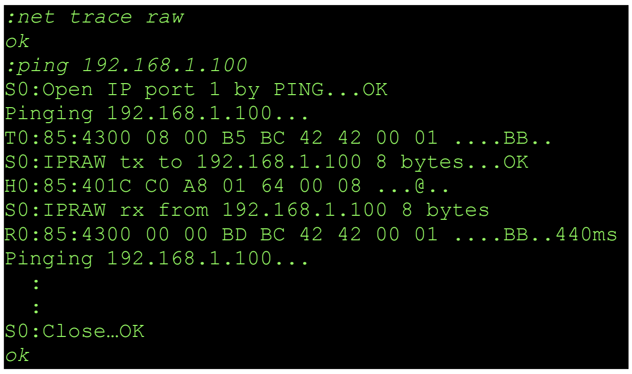
:PING Opening socket 0
Normal :PING ouput
Socket 0 sending packet to 192.1681.1.100
Socket 0 has received packet from 192.168.1.100
Repeats 5 times…

Time is longer because of time taken to print trace

Lines beginning with `S0` indicate that it is socket 0 that is being used. IPRAW just indicates that the socket is being used in the “IP raw” mode, which is what **:PING** uses to send an “ICMP Echo” (ping) packet. The socket trace lines in this case just show how many bytes are being sent and received, and from/to whom.

Raw Mode

If we turn on raw trace mode, we also get the actual bytes in the ICMP Echo packets:



```
:net trace raw
ok
:ping 192.168.1.100
S0:Open IP port 1 by PING...OK
Pinging 192.168.1.100...
T0:85:4300 08 00 B5 BC 42 42 00 01 ....BB..
S0:IPRAW tx to 192.168.1.100 8 bytes...OK
H0:85:401C C0 A8 01 64 00 08 ...@..
S0:IPRAW rx from 192.168.1.100 8 bytes
R0:85:4300 00 00 BD BC 42 42 00 01 ....BB..440ms
Pinging 192.168.1.100...
  :
  :
S0:Close…OK
ok
```

:PING Opening socket 0
Normal :PING output
Transmitting bytes on socket 0
Socket 0 sending packet to 192.1681.1.100
Header received on socket 0
Socket 0 received packet from 192.168.1.100
Data bytes received on socket 0
Repeats 5 times…

The output is as before, but with additional lines. These additional lines begin with:

`T0:`	bytes Transmitted on socket 0
`R0:`	bytes Received on socket 0
`H0:`	Header bytes received on socket 0. Before data is received a “header” is received indicating the data size and source of the following bytes. The exact meaning of the header bytes is beyond the scope of this document - see the WIZNet W5300 chip specification (But in this case there are 4 bytes IP address (C0 A8 01 64 hex = 192.168.1.100 decimal), and 2 bytes length of following data, in this case 8.)

If you were to look at RFC 792 (ICMP specification) would see that the format of an Echo packet is:

1 byte	Type field
1 byte	Code field
2 bytes	Checksum field
2 bytes	Identifier field
2 bytes	Sequence number field

The meaning of these is outside the scope of this document, but you can see the 8 bytes both sent and received follow this format.

There is a CPU address shown before the raw bytes, and this is a segment:offset-in-page-1 address of the bytes in memory. The exact address shown is of little meaning but can indicate where the read or write has come from ie. whether it is application memory or EPNET's memory.

More Complicated Trace Example: :FTP LOGIN

With tracing off:

```
:net trace off
ok
:ftp login 192.168.1.100
User name [anonymous]:
Password [anon@localhost]:
```

With tracing on (40 column screen):

```
:net trace off
ok
:ftp login 192.168.1.100
FTP1:Connect to 192.168.1.100…
S1:Open TCP port 22 by FTP…OK
S1:Connect to 192.168.1.100:21…OK
OK
FTP1:Login...
S1:TCP rx 143 bytes
FTP1:Rx 220-FileZilla Server 0.9.53 beta
220-written by Tim Kosse (tim.kosse@file
zilla-project.org)
220-Please visit https://filezilla-proje
ct.org/
FTP1:code=220
User name [anonymous]:
FTP1: Tx USER anonymous
S1:TCP tx 16 bytes…OK
S1:TCP rx 37 bytes
FTP1:Rx 331 Password required for anonym
ous
FTP1:code=331
Password [anon@localhost]:
FTP1:Tx PASS anon@localhost
S1:TCP tx 21 bytes…OK
S1:TCP rx 15 bytes
FTP1:Rx 230 Logged on
FTP1:code=230
ok
```

EPNET's FTP protocol decides to open a connection
First opens a socket
And then connects to the server

EPNET's FTP protocol starts the login process
Socket 1 receives the welcome message from the server
Passes it to EPNET's FTP protocol

EPNET's FTP protocol has to scan the message for the
EPNET prompts for the user name
EPNET sends the FTP USER command to the server
Socket 1 sending the command
Socket 1 receives the server response
Passes it to EPNET's FTP protocol

EPNET's FTP protocol interprets the response
EPNET prompts for the password
EPNET's FTP protocol sends the FTP PASS command
Socket 1 sending the command
Socket 1 receives the server command
EPNET's FTP protocol receives the response
EPNET's FTP protocol interprets the response