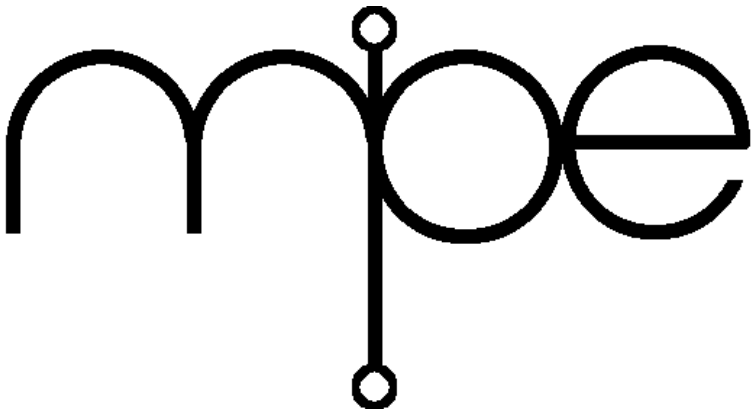
PowerNet TCP/IP Stack

v5.0

Stephen Pelc, Graham Stevenson



Published by Microprocessor Engineering

PowerNet TCP/IP Stack
User manual
Manual revision 5.0
18 July 2014

Software
Software version 5.0

For technical support
please contact your supplier

For further information
MicroProcessor Engineering Limited
133 Hill Lane
Southampton SO15 5AF
UK

Tel: +44 (0)23 8063 1441
Fax: +44 (0)23 8033 9691
e-mail: mpe@mpeforth.com
tech-support@mpeforth.com
web: www.mpeforth.com

Table of Contents

1 Introduction

1.1 What Do You Get?

PowerNet is a TCP/IP networking stack code written in Forth with support for:

1. Ethernet Layer. Reference driver for SMSC LAN91C9x series of Ethernet chips. Includes a packet sniffer to test reception and Ping to test transmit and receive. Additional Ethernet drivers can be found in the CPU-specific drivers directories supplied with your Forth cross-compiler.
2. SLIP Layer. Uses a serial Interface and multiple connections are supported
3. Routing Table. IP/Hardware routing table, with life-timers on routes.
4. IP Layers.
    a. ICMP handler. ECHO Request coded for PING, other codes have stubs.
    b. UDP handler. Unconnected data packets.
    c. DHCP client. Permits almost automatic configuration.
    d. DNS client. Converts URLs to IP addresses.
    e. SNTP client. Used to synchronise the local clock.
    f. TCP handler. Connected data streams.
5. Socket Layer. The sockets interface being used is based on BSD/Winsock
6. Services Layer
    a. Primitive TFTP client (Trivial File Transfer Protocol)
    b. Multi-threaded Telnet server for Forth Interpreter over TCP/IP
    c. Multi-threaded HTTP server with ASP and CGI.
    d. Modbus server template.
    e. Echo server
7. Examples
    a. Sending mail by SMTP.
    b. ModBus client and server templates.
    c. POST handler with application binary updater.
    d. Configuration tools.

1.2 Documentation

The main commentary on the code and all the glossaries are generated directly from the source by the DocGen utility supplied with MPEs VFX Forth for Windows. Both PDF and HTML versions are provided.

The code and its *DocGen* documentation should be considered the primary reference. Please read it.

1.3 Source Tree

```
PowerNet
|
+------ ARP                       Address Resolution Protocol
+------ DEFINES.FTH               Various IP Constants
+------ DHCP.FTH                  DHCP client
+------ DIAGS.FTH                 Diagnostic / Test routines
+------ DNS.fth                   DNS client
+------ GLOBALS.FTH               Global Vars/Buffers
+------ ICMP.FTH                  ICMP Packet handlers
+------ IP.FTH                    LowLevel IP access
+------ NETCODE.FTH               Network and CPU dependent code
+------ POWERNET.BLD              Stack Build File
+------ PRIMITIV.FTH              Useful code fragments
+------ ROUTING.FTH               Routing table handler
+-------SNTP.fth                  SNTP client.
+------ STRUCTS.FTH               Various IP/BSD Structures
+------ TCP.FTH                   TCP Protocol Handler
+------ UDP.FTH                   UDP Protocol Handler
+------ USERVARS.FTH              Required System Variables
|
+------ ETHER
|       +------ ETHTASK.FTH      Ethernet TxRx Dispatcher Task
|       +------ SMC91C9X.FTH     Reference Ethernet Driver
|
+------ SLIP
|       +------ SLIPCOM.FTH      SLIP interface
|
|------ EXAMPLES
|       +------- SMTPmail        SMTP mail examples
|       +------- WebConfig.fth   Application Configuration
|       +------- WebPost.fth     POST and binary update
|       +------- MBusClnt.fth    Modbus framework
|
+------ SOCKETS
|       +------ BSD.FTH          BSD Sockets API
|       +------ SOCKPRIM.FTH     Low Level Socket struct manip.
|
+------ QUEUES
|       +------ PBUFFERS.FTH     Queue Buffer handlers
|       +------ QPRIMS.FTH       En/DeQueue code
|       +------ QUEUES.FTH       Describe/initialise queues
|       +------ RAM.FTH          Describe TxRx RAM Useage
|
+------ SERVICES
|       +------ HTTP.FTH         MultiThreaded Web Server
|       +------ MbusSrvr         ModBus Server framework
|       +------ Pages.fth        Example web page handlers
|       +------ Servers.fth      Server core code
|       +------ TcpEcho          TCP Echo Server
|       +------ TELNET.FTH       Multi-threaded Telnet Server
|       +------ TFTP.FTH         Simple TFTP Server
|
+------ TestPages                Example web pages
|
```

2 PowerNet.bld - primary build file

When all else fails, read the source code.

PowerNet contains a large number of confiration options. These are contained in *PNconfig.fth*. You should make copy in your application folder or use an existing configuration file.

2.1 Heap

PowerNet needs a heap. The bigger the heap, the more traffic PowerNet can handle. 32kb is good for initial testing if the number of free PBUFFS in PNconfig.fth (see below) is kept low. If you have too low a heap size or request too many resources, PowerNet will complain at start up.

```
$0000:8000 equ sizeofheap \ 0=no heap, nz=size of heap
  1 equ heap-diags?       \   true to include diagnostic code
```

2.2 Multitasker

The multitasker is required.

```
1 equ tasking?             \ true if multitasker needed
  6 cells equ tcb-size     \   for internal consistency check
  0 equ event-handler?     \   true to include event handler
  0 equ message-handler?   \   true to include message handler
  1 equ semaphores?        \ true to include semaphores
```

2.3 Other Forth equates

The equate `RP-SIZE` should be set to $0200 bytes. Powernet is particularly heavy in its use of locals and hence of the return stack.

2.4 Configuring the stack

First of all you must add a text macro to your main control file to tell it where the stack is located. The code below was used during development.

```
c" C:\buildkit.dev\software\AddOns\PowerNet\Dev"
               setmacro IpStack \ where the PowerNet stack lives
```

The macro `IPSTACK` tells the cross compiler where the file *POWERNET.BLD* is located.

By default, *PowerNet.bld* compiles configuration data from the file *PNconfig.fth*, which is always provided with PowerNet. If you need a different configuration, create a different file based on *PNconfig.fth*. The `EQU`ate `PNconfigured?` **must** be present and non-zero in your configuration file. It permits you to compile your configuration file before compiling *PowerNet.bld* and prevents compilation of the default configuration file.

```
create PNETver$ \ -- addr
```

PowerNet version string.

```
: .PNET         \ --
```

Display PowerNet version string.

2.5 Default console I/O for tasks

All PowerNet tasks should define an input/output device used for debugging. By default, this is `CONSOLE` as set up for the interactive Forth. If you need to change this, modify the word `ConsoleIO` below or provide a version before PowerNet is compiled.

```
: ConsoleIO     \ --
```

Define the default debug device for PowerNet tasks.

```
: >pos          \ n --
```

Move to position n in the output stream for `EMIT` and friends.
Later versions of the MPE kernel code use a simpler timing mechanism by default.

2.6 Default Ethernet and IP addresses

These are compiled if they have not been previously defined.

```
create EtherAddress     \ -- addr
```

Holds the Ethernet MAC address (six bytes). Note that you must obtain these from the IEEE (www.ieee.org) or from other sources. This definition is usually provided by the Ethernet driver which must be compiled first.

```
create IpAddress        \ -- addr
```

Holds the Ethernet IP address (four bytes). The range 192.168.xxx.yyy is commonly used for private networks. This definition is usually provided by the Ethernet driver which must be compiled first. The data is in network order.

```
create EnetIPMask       \ -- addr
```

IP mask for addresses on Ethernet port. The data is in network order.

```
create IPGateway        \ -- addr
```

Gateway attached to Ethernet port. The data is in network order. Set to 0.0.0.0 if there is no gateway.

2.7 Compiler extensions

These words will probably be removed in a future release.

```
: mask-ints     \ --
```

Disable interrupts. Needed with pre-emptive schedulers.

```
: unmask-ints   \ --
```

Re-enable interrupts. Needed with pre-emptive schedulers.

```
: flushDebug    \ -- ; SFP003
```

Flush the debug queue. Needed if debug is handled through some buffered serial lines, SLIP or Telnet.

2.8 Compiling PowerNet

This section of *POWERNET.BLD* pulls in all the required files. Note that if you have coded versions of the network order memory words and/or the Internet checksum routine (heavily used), you can use an existing (or create your own) for the CPU you are using. The file for CPU specific code is *NETCODE.FTH*. If you define an EQUATE `CPU=xxx` where xxx is one of ARM, 386, 68k and so on, a CPU-specific code definition will be compiled, otherwise a default high level (and slower) version will be used

2.9 Compile the required services

This section compiles the required services and establishes a task to handle any output they may need.

`: RunServices  \ -- ; task action`
This is the action of `SERVICETASK`, which provides I/O support for all the services to avoid blocking problems in the service tasks themselves.

`task ServiceTask  \ -- addr`
The task used to provide I/O for services.

`: RunServiceTask  \ --`
Run the service task.

2.10 Initialisation

Some initialisation code is provided in POWERNET.BLD to make it easier to manage extensions.

`: initip  \ -- ; initialise the data structures`
Initialise the PowerNet data structures and the multitasker.

`: iptasks  \ -- ; start required tasks`
Start all the required tasks.

`: PowerNet  \ --`
Start PowerNet.

`: .PowerNet  \ --`
Display PowerNet state.

2.11 Sanity checks

`Next-user @ up-size > [if]`
Check that the USER area is large enough.

3 PowerNet configuration

The file <*PNET*>*PNconfig.fth* contains the default configuration for PowerNet. To set your own configuration, copy <*PNET*>*PNconfig.fth* and rename it. You can put the file where you like, so placing it in your main application source folder is sensible. Then compile your file **before** *PowerNet.bld*, which will ignore the default file if it finds a previously loaded configuration.

3.1 Features and Services

The following group of equates defines the facilities to be compiled. The defaults here can be overridden by defining equates before the configuration file is compiled; this can be useful for testing configuration changes.

```
 1 equ ethernet?        \ -- n ; nz for Ethernet systems
```
True to include Ethernet support.

```
 0 equ slip?            \ -- n ; nz to include SLIP
```
True to include a SLIP handler on a serial port.

```
0 equ GenericIP?        \ -- flag
```
Set this equate true if the Generic IP device structure defined in *ETHERCOM.FTH* is required. This is only required for systems using multiple IP devices in future releases of PowerNet.

```
1 equ useDHCP?          \ -- flag
```
Set non-zero to compile DHCP client code. If you set this and want an IP address to be assigned by a server, set `IpAddress` to 0.0.0.0.

```
0 equ useDNS?           \ -- flag
```
Set non-zero to compile DNS client code.

```
1 equ DNSauto?          \ -- flag
```
If `useDNS?` and `useDHCP?` are non-zero, set this non-zero to get the DNS server address using DHCP.

```
0 equ DNSauto?          \ -- flag
```
`DNSauto?` defaults to false if the build does not include both *DHCP* and *DNS*.

```
0 equ useSNTP?          \ -- flag
```
Set non-zero to compile SNTP client code.

```
1 equ SNTPauto?         \ -- flag
```
If `useSNTP?` and `useDHCP?` are non-zero, set this non-zero to get the SNTP/NTP server address using DHCP.

```
[undefined] SNTPauto? [if]
```
`SNTPauto?` defaults to false if the build does not include both *DHCP* and *SNTP*

```
0 equ RAMconfig?        \ -- flag
```
This word is now redundant, but is left in for compatibility with earlier systems. Its function is replaced by the tools in *Examples**WebConfig.fth*. Some systems use dynamic configuration, either because of DHCP or because code is run from Flash and the configuarion is loaded from (say) serial EEPROM at runtime. For these systems, set this equate non-zero so that some data buffers are created in IDATA or UDATA space rather than CDATA space. Sample production configuration routines can be found in *Examples**WebConfig.fth*. These rely on the words `RAM>DataFlash` and `DataFlash>RAM`. Note that systems that copy Flash to RAM for execution can leave this equate set to zero.

```
 0 equ tftp?            \ -- n
```
True to include TFTP.

```
 1 equ tcp?             \ -- n
```
True for TCP as well as UDP. UDP is always compiled.

```
1 equ Servers?
```
True if any servers are required, e.g. HTTP or Telnet.

```
1 equ Clients?
```
True if client code is required, e.g. HTTP or Telnet.

```
1 equ http?             \ -- n ; nz to include HTTP
```
True to include the HTTP server. Note that TCP and server support are required.

```
1 equ MemPages? \ -- n
```
Set this non-zero to compile code for HTTP pages to be served from memory, e.g. Flash.

```
0 equ FilePages?        \ -- n
```
Set this non-zero to compile code for HTTP pages to be served from a *FATfiler* file system.

```
1 equ telnet?           \ -- n ; nz to include Telnet
```
True to include the Telnet server. Note that TCP and server support are required.

```
0 equ ftp?              \ -- n ; nz to include Telnet
```
True to include the FTP server. Note that TCP and server support are required.

```
0 equ echo?             \ -- n ; nz to include Echo
```
True to include the Echo server. TCP and server support are required.

```
0 equ snmp?             \ -- n ; nz to include SNMP
```
True to include SNMP.

```
0 equ smtp?             \ -- n ; nz to include SMTP
```
True to include SMTP.

3.2 Diagnostics

The next group of equates controls the generation of debugging information.

```
1 equ diags?            \ -- n
```
Set true to include diagnostic code (recommended).

```
1 equ XC?               \ -- n
```
True if compiled by cross compiler. False if compiled by a hosted system such as VFX Forth for DOS.

```
0 equ tcpdebug?         \ -- n
```
True for lots of TCP debug info.

```
0 equ ICMPmon?          \ -- n
```
True to monitor ICMP packets.

3.3 Queues and Buffers

```
#1520 equ PSIZE         \ -- n
```
Data size in each PBuf, including all transport headers. This value should be cell aligned. Ethernet packets require 1518 bytes from the start of the Ethernet header to the end of the Frame Check Sequence (FCS). Note that some Ethernet DMA hardware does transfer the FCS.

```
#10 equ BUFFHDRSIZE     \ -- n
```
header size; see *Queues\PBuffers.fth* for details.

```
#4 equ  NUMFREEPBUFS    \ -- n
```
Number of PBUFs. This is the maximum number of packets that can be "in transit" in PowerNet at any one time. The PBUFs are allocated from the heap when PowerNet starts, so there must be enough heap space to contain the buffers, otherwise you will see a console error message.

```
1 equ #TXpbufs          \ -- n
```
High reliability systems, and especially internet-facing devices should set this equate to 1. This reserves an extra PBUF for TCP transmission, which guarantees that transmission can proceed even if the other `NUMFREEPBUFS` PBUFs are consumed by incoming data packets, as can happen when `NUMFREEPBUFS` is small, a number of sockets are open and/or a task is sending a quantity of data on a single socket without checking for input. Setting `#TXpbufs` to one can be particularly important when using Telnet over lossy links - geologists please note!

```
#10 equ TRXBUFFSIZE     \ -- n
```
Data size in each IO buffer (excl. hdr)

```
TRXBUFFSIZE BUFFHDRSIZE + equ /TRxBuff  \ -- n
```
Size of complete IO buffer structure

```
#40 equ NUMTRXBUFFERS   \ -- n
```
Number of IO buffers

```
/TRxBuff NUMTRXBUFFERS * equ /TRxBuffers        \ -- n
```
size of TRxBuffers
Do not change these unless you are really sure you know what you are doing.

```
$F0 equ DEV_MASK        \ -- mask
```
Router device mask (8 bits). The device type is contained in the upper four bits and the device number within a type is contained in the lower four bits.

```
$E0 equ ETHER_MASK      \ -- mask
```
Ethernet ports are in the range $E1..EF.

```
$00 equ SLIP_MASK       \ -- mask
```
SLIP ports are in the range $01..0F.

```
$E1 equ ETHER_PORT      \ -- n
```
Port identifiers for Ethernet devices are in the range $E1..EF

3.4 Routing

```
#32 equ MAX_IPADDRS     \ -- n
```
No of entries in the routing table (not dynamic yet).

```
#7200000 equ ROUTE_LIFE \ -- ms
```
Two hours in milliseconds.

```
#1000 equ ROUTE_SAMPLE_MS       \ -- ms
```
How often routes are tested im milliseconds.

```
#2000 equ ROUTE_SEARCH_TIME     \ -- ms
```
Maximum time allowed for route searches.

3.5 ICMP

```
3 equ ARP_REQ_TRIES     \ -- n
```
Number of ARP request tries before failure.

3.6 IP

```
0 equ PacketTask?       \ -- n
```
Set this non-zero to use a separate task to handle incoming IP packets. This uses more code and RAM, but may give better performance on some systems. The default is 0.

3.7 TCP configuration

```
#1460 equ TCPDATASIZE   \ -- n
```
Transmit buffer size of a PBUF less the standard IP and TCP headers. The largest value for Ethernet is 1460.

```
#1460 2 * equ TCPTXBUFFSIZE     \ -- n
```
Size of the retransmission buffer for a TCP socket. One buffer of this size is allocated from the heap when a TCP socket is created. If you have enough heap space, make this at least twice `TCPDATASIZE`.

```
#1460 equ TCPWINDOWSIZE \ -- n
```
TCP window size.

```
0 equ genWinSize?       \ -- u
```
Selects the strategy used to generate the TCP window size advertised by PowerNet.
PowerNet queues input packets in the socket structure before passing them to the BSD layer. Usually, ACKs are delayed. There are three ways to implement receive window handling, selected by the equate `genWinSize?`.

- `genWinSize?`=0. The receive window is always set to `TCPWINDOWSIZE`.
- `genWinSize?`=1. By default, we assume that if all the input packets have not been consumed by the time the ACK is sent, the system is under heavy load and we do not want any more input for the moment. This is indicated by setting the window size to zero. Otherwise, the receive window is set to `TCPWINDOWSIZE`. Although this "bang bang" approach is very crude, it works well unless many packets are being sent from PowerNet, as can happen when web pages are served from memory.
- `genWinSize?`=2. The word `genWindowSize ( *sk *tcp -- )` is provided by you to suit your application. See *tcp.fth* for the examples.

```
#1460 equ defMSS        \ -- len
```
Define the maximum segment size (packet size less transport, IP and TCP headers). On a LAN 1460 is the default. For WAN and broadband access, lower values may be appropriate, e.g. 1420. The values sent in the SYN and SYN/ACK packets may be modified by intermediate routers.

```
#5 tick-ms max equ TcpIdleMs    \ -- ms
```
Interval in miliseconds between TCP socket polls. Using this timer mechanism reduces the CPU load at the possible expense of transmission performance (by 10-20%) on local area networks. Setting the value of `TcpIdleMs` to zero turns the load reduction mechanism off.
Note that on some systems, setting `TcpIdleMs` non-zero may improve performance because of the reduction in polling time. For tuning, we recommend an initial value of 5, with the ticker rate (usually defined by `TICK-MS` in the main control file) set to 1 millisecond.

```
#100 equ TXDELAYTIME    \ -- ms
```
timer for delayed transmit (mSecs), default is 100 milliseconds.

```
#10 equ ACKDELAYTIME    \ -- ms
```
timer for delayed ack (mSecs), default is 10 milliseconds.

```
#5000 equ TXRETRYTIME   \ -- ms
```
timer for transmit retries (mSecs), default is 5 seconds.

```
#12 equ TCPMAXRETRIES   \ -- n
```
Maximum number of transmission retries before returning `SOCKET_ERROR`; the default value is 12. If a receiving socket fails to accept new data, the total timeout will be `TXRETRYTIME * TCPMAXRETRIES`, the default being 60 seconds.

```
#30000 equ TCPCONNECTTIME       \ -- ms
```
Timer for incoming connections to complete (mSecs); the default is 30 seconds.

```
#30000 equ TCPMSLTIME   \ -- ms
```
Timer for maximum segment lifetime (mSecs); the default is 30 seconds.

```
#7200000 equ TCPIDLETIME        \ -- ms
```
timer for idle disconnection (mSecs, default is 2 hours).

```
#80 equ TcpRstTime      \ -- ms
```
Delay before closing socket after a reset.

3.8 Sockets

```
#16 equ MAXSOCKETS      \ -- n
```
The maximum number of sockets available in the system. The socket data structures are allocated from the heap when required. The amount of data used for each socket depends on

- whether it is a UDP or TCP socket (more for TCP),
- whether it is a server socket that requires a service data structure. See the chapter on the server architecture for more details of this.

Do not set `MAXSOCKETS` much higher than you need under the expected maximum load otherwise overall system response may suffer. Note that within PowerNet, a socket number is a 16 bit item, so the range is 2..65535. Socket 1 is a special case.

Sockets are numbered from 1 to `MAXSOCKETS` and socket 1 is special. In practice, you have `MAXSOCKETS`-1 sockets to use.

```
#6000 equ FirstPortNumber       \ -- n
```
First ephemeral port number.

```
#9000 equ LastPortNumber        \ -- n
```
Last+1 ephemeral port number.

3.9 DNS client

```
1 equ DNSdebug? \ -- x
```
Set this non-zero to display DNS debug information during execution.

```
#53 equ DNSport#        \ -- port#
```
The standard port number for DNS servers.

```
3 equ #DNS              \ -- u
```
Maximum number of DNS attempts if no response.

```
5000 equ /DNSms \ -- ms
```
Maximum time (in milliseconds) allowed for a DNS transaction.

```
#512 equ /DNS           \ -- u
```
Maximum size of the DNS payload.

```
variable DNSserver      \ -- addr
```
Holds DNS server or 0 if not configured yet.

3.10 Servers and Services

`5 equ /ListenQ  \ -- +n`
TCP listening ports maintain a list of connections. This equate defines the number of outstanding connection attempts that can be queued. The default is 5, which is common for Unices such as Solaris, where it is referred to as the backlog value. Many browsers request up to four connections for each page and queueing connections reduces peak RAM load in the heap. **Do not** reduce this value to zero.

`0 equ SVlowRAM? \ -- x`
Set this non-zero to use the low RAM configuration for servers. Setting this non-zero increases the CPU load and may cause a significant increase in the number of small packets transmitted.

`0 equ SVsingle? \ -- x`
When non zero, the HTTP and Telnet servers accept only a single connection at at a time. This connection is run from the listening task, so reducing RAM usage and page serve performance. For applications on single-chip systems with limited RAM and in which the HTTP and Telnet servers are used only for configuration and maintenance by humans, this is a useful setting, especially where the application's RAM needs are high.

`#80 equ HTTPPort#       \ -- n ; standard is 80`
Define the port used for the HTTP server. The standard port is 80.

`#15 #1000 * equ HTTP_KEEPALIVE_TIME     \ -- ms`
The keep alive time in milliseconds used by HTTP.

`#7 equ EchoPort#        \ -- n`
Port on which Echo server listens.

`#5023 equ TelnetPort#   \ -- n ; standard is 23`
Define the port used for the Telnet server. The standard port is 23, but 5023 is the default set for PowerNet as most application Telnet servers are private.

`#21 equ FTPcmdPort#     \ -- u ; standard is 21`
Define the command port used for the FTP server. The standard port is 21.

`#20 equ FTPdataPort#    \ -- u ; standard is 20`
Define the default data port used by the FTP server. The standard data port is 20.

`#20 #1000 * equ FTP_DATA_ESTAB_MS       \ -- ms`
Time in milliseconds that an FTP transfer command such as RETR will wait for a passive mode connection.

`2 equ #FTPmaxConns      \ -- u`
If you select multiple files to transfer in an FTP client, the client may well open one connection per file. Since each connection requires a task, service area, and two sockets, the system can easily run out out of heap space and/or sockets. See `sizeofheap` in the control file and `MAXSOCKETS` in this file.

3.11 End of configuration

This must be the last section of the configuration file.

`1 equ PNconfigured?     \ -- n`
Must be defined non-zero after all configuration information has been defined.

4 Debugging tools

4.1 Miscellaneous

`: ?dump \ addr len --`
Now redundant, but does nothing for "addr 0 DUMP".

`: .decimal \ n --`
display a value in decimal.

`: .hex \ n --`
display a value in hexadecimal.

4.2 Using the system console

`: consoleIO \ --`
Select debug console for output. By default this is the CONSOLE device.

`: [con \ -- ; R: -- ipvec opvec`
Save the current I/O devices on the return stack, and set I/O to the console. Restore the previous state with `io]`.

```
[con ... io]
```

`: con] \ -- ; R: ipvec opvec --`
Restore the I/O devices from the return stack.

`: .ConLine \ caddr len --`
Display text with leading CR on Forth console.

`: debug_emit \ char --`
As EMIT but always uses the CONSOLE device.

`: debug_cr \ --`
As CR but always uses the CONSOLE device.

`: debug_space \ --`
As SPACE but always uses the CONSOLE device.

`: debug_spaces \ n --`
As SPACES but always uses the CONSOLE device.

`: debug_>pos \ n --`
Position the console output to column n.

`: debug_type \ addr len --`
As TYPE but always uses the CONSOLE device.

`: debug_count&type \ c-addr --`
As COUNT TYPE but always uses the CONSOLE device.

`: debug_. \ n --`
As . but always uses the CONSOLE device.

`: debug_.hex \ n --`
As .HEX but always uses the CONSOLE device.

`: debug_.decimal \ n --`
As .DECIMAL but always uses the CONSOLE device.

`: debug_.lword \ n --`

As .LWORD but always uses the CONSOLE device.

```
: debug_.byte   \ n --
```

As .BYTE but always uses the CONSOLE device.

```
: debug_.ASCII  \ char --
```

As .ASCII but always uses the CONSOLE device.

```
: debug_.s      \ --
```

As .S but always uses the CONSOLE device.

```
: debug_dump    \ addr len --
```

As DUMP but always uses the CONSOLE device.

4.3 Stack checking

```
: ?StackEmpty   \ --
```

If the stack depth is non-zero, issue a console warning message and clear the stack. This is often used at the end of a task loop.

```
: DictTop  here  ;
```

Used by some debug tools as the highest permitted address in the dictionary. If not previously defined, `HERE` is used.

```
: name?         \ addr -- flag                                  MPE.0000
```

Check to see if the supplied address is a valid NFA, returning true if the address appears to be a valid NFA. This word is implementation dependent. For MPE cross compilers, a valid NFA for MPE embedded systems satisfies the following:

- All characters within string are printable ASCII within range 33..126
- String Length is non-zero in range 1..31 and bit 7 is set, ignore bits 6, 5

```
: ip>nfa        \ addr -- nfa
```

Attempt to move backwards from an address within a definition to the relevant NFA.

```
: SF{           \ n -- ; R: -- depth
```

You can check for stack faults with:

```
 n SF{ .... }SF
```

where n describes the stack change between SF{ and }SF. If the stack change is different, an error message is generated. This word will work on most systems in which the return address is held on the return stack.

```
: }SF           \ -- ; R: depth -- ; perform stack check
```

The end of an SF{ ... }SF structure. This word is not strictly portable as it assumes that the Forth return stack holds a valid return address. In the vast majority of cases the assumption is true, but beware of some 8051 implementations. See SF{

```
: rdepth        \ -- n
```

Return the number of items on the return stack.

```
: .rs           \ --
```

Display to the console the current contents of the return stack. Where possible a word name is also displayed with the data value.

4.4 Cold Chain

`: showExecChain \ item --`
Show the contents of a chain, given an item in the chain.

`: showColdChain \ --`
Display the cold chain.

5 Network operations - CPU specific

The file NETCODE.FTH contains code that will show significant benefit from being coded, or is sensitive to the byte order of the underlying CPU.

The code in this file is used by PowerNet v3 onwards, and should only be compiled once if both PowerNet and NETBOOT.FTH are both compiled.

5.1 Network order (big-endian) operations

Note that these functions have to be capable of fetching 32 bit cells from 16 bit aligned addresses, not just from 32 bit aligned addresses.

Note also that these routines assume a byte-addressed CPU.

`: w@(n)  \ addr -- u16`
Network order 16 bit fetch.

`: w!(n)  \ u16 addr --`
Network order 16 bit store. Must write MSB first if writing to some hardware

`: @(n)  \ addr -- u32`
Network order 32 bit fetch.

`: !(n)  \ u32 addr --`
Network order 32 bit store.

`: w,(n)  \ w --`
Network order W,

`: ,(n)  \ x --`
Network order version of , (comma).

`: w,(n)  \ w --`
Network order W,

`: ,(n)  \ x --`
Network order version of , (comma).

5.2 Internet checksum

Because this code is used so much in the system, it is worth coding it for maximum efficiency. One of the following equates may be defined (any value) in the control file to compile CPU specific code for the Internet checksum:

`CPU=Cortex`
: Cortex v7, e.g. M3, but not Cortex-M0.

`CPU=ARM` All ARM variants using the 32 bit ISA

`CPU=H8/300H`
: Hitachi H8/300H and H8S

`CPU=68K` Motorola 68xxx CPUs

`CPU=386` Intel 386 and above (i32)

If no CPU specific equate is defined, the high level code will be used.

If the checksum words have been defined previously, this section of code is ignored.

`[defined] CPU=Cortex [if]`
If this equate is defined (any value will do), a Cortex (not M0) machine code IP checksum routine will be used.

`code (cksum)    \ cksum addr len -- cksum'`
ARM IP Checksum routine. This word is safe against alignment issues as the network order word fetch is done as two byte fetches.

`[defined] CPU=ARM [if]`
If this equate is defined (any value will do), an ARM machine code IP checksum routine will be used.

`code (cksum)    \ cksum addr len -- cksum'`
ARM IP Checksum routine. This word is safe against alignment issues as the network order word fetch is done as two byte fetches.

`[defined] CPU=H8/300H [if]`
If this equate is defined (any value will do), an H8/300H or H8S machine code IP checksum routine will be used.

`[defined] CPU=68K [defined] CPU=Coldfire  or [if]`
If this equate is defined (any value will do), a 68xxx/Coldfire machine code IP checksum routine will be used.

`[defined] CPU=386 [if]`
If this equate is defined (any value will do), a i386+ machine code IP checksum routine will be used. N.B. This code has not been tested!

`: (cksum)       \ cksum addr len -- cksum'`
Compute the partial internet checksum of a memory area. This is a high level implementation that can be used by any CPU. The word (CKSUM) is useful when calculating header checksums, and avoids the need to create the pseudoheaders described in the TCP/IP literature.

`: cksum         \ addr len -- cksum ; computed as 16 bit ints`
Compute the internet checksum of a memory area. This a high level implementation that can be used by any CPU.

`STRUCT /pseudohdr       \ -- size`
The pseudoheader structure for checksum calculation. This word is for illustration and is **not** compiled.

`: PHcksum       \ *hdr len ipsrc ipdest ipproto -- cksum`
A generic TCP/UDP checksum function. The parameters supplied are used to form a 'pseudo-header' from which the acual checksum is generated or checked.

6 Extra USER variables

6.1 Common to all tasks

```
cell +User SocketErrorCode      \ -- addr
```
Last socket error.

```
cell +User my_hsocket   \ -- addr
```
Socket# owned by this task.

```
cell +User my_socket    \ -- addr
```
Socket address owned by this task

```
\ cell +user *emitBuff  \ -- addr
```
Start of the current emit buffer

```
\ cell +user *emitNext  \ -- addr
```
Pointer for next buffer location

```
\ cell +user *emitQueue \ -- addr
```
Pointer to interface emit queue

```
\ cell +user emitmode   \ -- addr
```
0=char, 1=block, 2=udp socket

```
\ cell +user *keyBuff   \ -- addr
```
Contains current key buffer address

```
\ cell +user *keyNext   \ -- addr
```
Pointer to next byte in buffer

```
\ cell +user keyCount   \ -- addr
```
count of bytes left in buffer

```
\ cell +user *keyQueue  \ -- addr
```
pointer to interface key queue

6.2 SLIP variables

```
cell +User portid       \ -- addr
```
SCC port identifier 01-0n

```
cell +User rxstate      \ -- addr
```
current rx packet state

```
cell +User *rxpbuf      \ -- addr
```
pointer to current rx pbuf

```
cell +User *rxnow       \ -- addr
```
pointer to next rx char

```
cell +User rxcount      \ -- addr
```
bytes entered into buffer

```
cell +User rxspace      \ -- addr
```
space left in buffer

```
cell +User *txpbuf      \ -- addr
```
pointer to current rx pbuf

```
cell +User *txnow       \ -- addr
```
pointer to next tx char

```
cell +User txcount      \ -- addr
```
bytes entered into buffer

```
cell +User txspace      \ -- addr
```
space left in buffer

```
cell +User RXExpSlip    \ -- addr
```
receive expiry time

```
cell +User *community   \ -- addr
```
pointer to community string

```
cell +User commlen      \ -- addr
```
length of community string

```
cell +User pdutype      \ -- addr
```
pdu type (Get etc.)

```
cell +User reqid        \ -- addr
```
pdu request identifier

```
cell +User errStatus    \ -- addr
```
pdu error status

```
cell +User errIndex     \ -- addr
```
pdu error index

```
cell +User numVars      \ -- addr
```
vars requested in pdu

```
cell +User *tagsIn      \ -- addr
```
pdu tags in pointer

```
cell +User *varsOut     \ -- addr
```
pdu vars out pointer

```
cell +User *nextVar     \ -- addr
```
pointer to next variable

```
cell +User *outPacket   \ -- addr
```
pointer to outgoing packet buffer

```
cell +User *replystart  \ -- addr
```
start of response

```
cell +User replylen     \ -- addr
```
length of response

```
cell +User *data        \ -- addr
```
current data pointer

```
cell +User spaceleft    \ -- addr
```
remaining length of data buffer

```
cell +User taddr        \ -- addr
```
temp data pointer

```
cell +User trapgen      \ -- addr
```

generic trap identifier

```
cell +User trapspec     \ -- addr
```

specific trap identifier

7 Queues

The PowerNet `QUEUE` data structure is the head of a linked list of queue buffer data structures, usually a `PBUF` or an I/O buffer. The first cell of anything that is added to a `QUEUE` **must** have a link field at offset 0, the first cell.

7.1 QUEUE structure

```
: QLock         \ -- ; lock out interrupts
```
If QUEUEs are used in interrupts, or if the scheduler is interrupt driven, place your locking code here. Do not forget to change the `COMPILER ... TARGET` macros too.

```
: QUnLock       \ -- ; reenable interrupts
```
If QUEUEs are used in interrupts, or if the scheduler is interrupt driven, place your locking code here. Do not forget to change the `COMPILER ... TARGET` macros too.

```
: PeekQ         \ *queue -- *head
```
Return the first item in the queue without removing it. Useful for testing and handling partially full/empty buffers.

```
: Dequeue       \ *queue -- *head|0 ; 0 indicates nothing to remove
```
Remove the next item from the queue.

```
: Enqueue       \ *buffer *queue --
```
Add buffer to queue.

```
: InitQ         \ *queue --
```
Initialise/reset the queue.

8 QUEUE and buffer allocation

Create the various IO queues needed

```
0 buffer: FirstQ        \ -- addr
```
Address of first location in the queue RAM area. Used by initialisation code.

```
queue buffer: FreePBufQ \ -- addr
```
Free PBuffers ready for RX & TX

```
  queue buffer: IPinQ   \ -- addr
```
Holds unprocessed incoming packets.

```
  queue buffer: TxEnetQ \ -- addr
```
outgoing ethernet data buffers

```
  queue buffer: TxSlipQ \ -- addr
```
outgoing Slip data buffer

```
0 buffer: LastQ         \ -- addr
```
Address of last location +1 in the queue RAM area. Used by initialisation code.

```
/QBarea buffer: FirstQ  \ -- addr
```
Address of first location in the queue RAM area. Used by initialisation code.

```
/TRxBuffers buffer: TRxBuffers  \ -- addr
```
The block of I/O buffers.

9 PBUF buffers

9.1 Introduction

PBUFs are used to hold data which is received or transmitted as IP packets. Rather than copy data repeatedly, the system passes pointers to these structures. In order to avoid heavy heap use, these buffers are preallocated.

Collections of buffers are handled by the QUEUE data structures.

The function of PBUFs is to hold data that will be received or transmitted to a device. Headers are added to application data. The worst case occurs for Ethernet systems:

```
Ethernet header
IP header
TCP, UDP or other protocol header(s)
Application data
```

Traditionally, TCP/IP stacks based on the BSD implementations use separate buffers for each data area. In order to avoid additional heap allocations and data copying, PowerNet uses a single buffer and moves pointers as the PBUF is moved up and down the TCP/IP stack.

9.2 Data structures

`struct pbuf_hdr \ -- size ; size of pbuf_hdr structure`
PBUF buffer header definition. The first four fields (10 bytes) are common with the obsolete BF_HDR definition.

`PSIZE pbuf_hdr + equ pbuf \ -- n`
Size of complete pbuf structure.

`$3FFF equ BUFFSIZEMASK \ -- mask`
The buffer size is held in the low 14 bits of the 16 bit type field. This permits a maximum data size of `BUFFSIZEMASK` bytes.

`$C000 equ BUFFTYPEMASK \ -- mask`
The buffer type is held in the top 2 bits of the 16 bit type field.

`$0000 equ DATABUFFER \ -- n`
Set into the type bits for I/O buffers.

`$4000 equ PBUFFER \ -- n`
Set into the type bits for PBUFs.

9.3 PBUF handling

`: !pb_size \ n *pb -- ; set buffer size`
Set the size of a buffer and mark it as a PBUF.

`: PBHdrStart \ *pbuf -- addr ; of packet header`
Return the address of the current header.

`: PBHdrLen \ *pbuf -- len ; of packet header`
Return the length of the current header.

`: PBDataStart  \ *pbuf -- addr ; of data`
Return the address of the current data.

`: PBDataLen  \ *pbuf -- len ; of data`
Return the length of the current data.

`: !PBDataLen  \ len *pbuf -- ; of data`
Set the length of the current data.

`: !*PBData  \ offset addr --`
Set offset to buffer data.

`: bumpPBuffData  \ n *pbuf --`
Moves the data pointer/count along the buffer, incrementing the address, decrementing the count.

`: unbumpPBuffData  \ n *pbuf --`
Moves the data pointer/count back up the buffer, decrementing the address, incrementing the count.

9.4 Queue buffer allocation and release

`: PBinit  \ *pb --`
Initialise a PBUF.

`: AllocPbuf  \ -- *pb|0`
`ALLOCATE` and initialise a PBUF from the heap and return the PBUF address. If the buffer cannot be allocated, 0 is returned without waiting.

`0 value TxPbuf  \ -- addr`
Points to the dedicated transmit PBUF.

`0 value NextTxPbuf  \ -- addr`
Holds `TxPbuf` when the transmit PBUF is available, or zero when not available.

`: InitIOQueues  \ --`
Initialise the predefined queues and their I/O buffers and PBUFs.

`: GetPbuf  \ -- *pb|0`
Get a PBUF from the free PBUF queue and return the PBUF address. If there are no free buffers, 0 is returned without waiting.

`: GetTxPbuf  \ -- *pb|0`
Get a PBUF from the free PBUF queue and return the PBUF address. If there are no free buffers, try the dedicated transmit PBUF.

`0 equ QBdiags?  \ -- n`
When this equate is set non-zero, additional test code is compiled and included in `FREEQB` below.

`variable qbug  \ -- addr`
When set to non-zero, `FREEQB` will display diagnostics if the buffer is bad.

`PBUFFER PSIZE or equ PBcheck  \ -- x`
Check value for a PBUF.

`: FreeQB  \ *pb/bf --`
Place a PBUF or I/O buffer on the relevant free queue. N.B. `FreeQB` cleans up the buffers before requeueing them.

`: FreeQB  \ *pb/bf --`

Place a PBUF or I/O buffer on the relevant free queue. N.B. `FreeQB` cleans up the buffers before requeueing them.

10 Queue diagnostic routines

The code in *QUEUES/Qdiags.fth* is only compiled if the equate `DIAGS?` is non-zero.

`queue aligned equ /qdhead  \ -- len`
The head of the list is the original queue header forced to an aligned length.

`cell BUFFHDRSIZE + aligned equ /qdentry \ -- len`
Each entry in the list consists of a link field, plus the original PBUF header. The entry must be aligned.

`: .q  \ *queue --`
Display queue info.

`: @qlen  \ *queue -- len`
Count queue length.

`: .qlen  \ *queue txtaddr --`
Display the queue length.

`: .qs  \ --`
Display the lengths of the FreeIoQ and the FreePbufQ.

11 System wide equates

This chapter documents *DEFINES.FTH* which contains definitions of constants and equates used throughout PowerNet.

11.1 Application definitions

```
0 equ INVALID_SOCKET    \ -- 0
```

Words returning a socket number return zero to indicate an invalid socket.

```
-1 equ SOCKET_ERROR     \ -- -1
```

Many BSD layer and lower layers return -1 for an error because a return of 0 as a length is valid. It also permits systems to use `0<` as an error check. **Do not change**.

11.2 Standard TCP/IP and Winsock values

PowerNet provides a number of standard equates taken from TCP/IP definitions and the Winsock API. These are not documented here. If you are interested see *DEFINES.FTH*.

Although a large number of these equates are defined, and are not currently used, they occupy no memory in the target system because these are `EQU`ates rather than `CONSTANT`s.

12 TCP/IP data structures

The definitions are in the file *STRUCTS.FTH*.

You will find the books TCP/IP Illustrated, Volumes 1 and 2 useful when exploring the PowerNet data structures.

12.1 Primary structures

These structures are used in almost all systems.

`STRUCT ether_hdr \ -- size ; of ether_hdr structure`
Ethernet header. The fields have the prefix "ETHER_".

`STRUCT arp_hdr \ -- size ; of arp_hdr structure`
ARP header. The fields have the prefix "AR_".

`STRUCT arp_ip_data \ -- size ; of arp_ip_data structure`
ARP data.

`STRUCT icmp_hdr \ -- size ; of icmp header structure`
ICMP header structure. The fields have the prefix "ICMP_".

`STRUCT ip_hdr \ -- size`
IP header structure. The fields have the prefix "IP_".

`STRUCT udp_hdr \ -- size ; of udp header structure`
UDP header structure. The fields have the prefix "UH_".

`STRUCT (socket) \ -- size ; of (socket) structure`
The main socket structure. The fields have the prefix "SO_".

`STRUCT tcpcb \ -- size`
TCP control block structure. Notes:

- Sequence and ACK numbers are unsigned 32 bit numbers that can wrap. They **must** be compared using circular arithmentic.

`STRUCT port_struct \ -- size ; of structure`
Port control data. The fields have the prefix "PORT_". Unused by systems with only a single port.

`STRUCT iproute_struct \ -- size ; of structure`
Routing table entry. The fields have the prefix "IPROUTE_". The entries are added to when a new IP address is received.

`PSIZE ether_hdr - ip_hdr - udp_hdr - equ PDATA_MAX`
The size of the largest data block.

`STRUCT SOCKADDR_IN \ -- size`
Used to hold socket family (always AF_INET), port number and IP address. The fields have the prefix "SIN_".

12.2 SNMP structures

`STRUCT TAGLIST_ENTRY \ -- size`
SNMP Tx/Rx taglist entry definition. The fields have the prefix "SNMP_".

13 Helpers and primitives

`: AddLink \ item anchor --`
Used when a new item in the chain already exists, e.g. it has been `ALLOCATE`d. The item is added to the chain. Note that this word requires the link to be at offset 0 in the item being added.

`: link, \ var-addr -- ; lay a link in a chain whose head is at var-addr`
Add a link to a chain anchored at address var-addr. The old contents of var-addr are added to the dictionary as the new link, and the address of the new link is placed at var-addr. INTERPRETER only.

`: AddEndLink \ item anchor --`
Add an *item* (a structure) to the end of of the chain anchored at *anchor*. The link field must be at offset 0 in *item*.

`: DelLink \ item anchor -- ; remove item from chain`
Delete/Remove an item from a chain achored at address anchor. Note that this word requires the link to be at offset 0 in the item being removed.

`: ExecChain \ anchor --`
Execute the contents of a chain with the following structure:

```
link | xt | ...
```

Each word that is run has the stack effect

```
link -- link
```

Where link is the address of the link field in the structure. Thus, data that follows the xt can easily be accessed.

`: ShowPacket \ *pbuf -- ; display the packet`
A diagnostic.

`: >inet_aton \ caddr len -- ipaddr`
Convert a dotted Internet address string, e.g. 192.168.0.1, into an IPv4 address. If the string cannot be converted, `INADDR_NONE` (-1) is returned.

`: (>inet_digit) \ n -- n>>8`
Perform number conversion of the low byte, and shift *n* right by 8 bits.

`: >inet_ntoa \ ipaddr -- c-addr len`
Convert an IP address into a text string c-addr/len in dotted quad notation aaa.bbb.ccc.ddd (the standard text form).

`: .IPaddress \ ipaddr --`
Display a v4 IP address in dotted quad notation. Now obsolete.

`: .IPv4 \ ipaddr --`
Display a v4 IP address *ipaddr* in dotted quad notation.

`: .IPnet \ addr --`
Display the v4 IP address stored in network order at *addr*.

`: .IPloc \ addr --`
Display the v4 IP address stored in native/local order at *addr*.

`: ?free \ addr|0 --`

Perform `FREE DROP` if the input is non-zero.

`: CopyEthAdd \ src dest -- ; copy ethernet address`
Copy an Ethernet address. This is an ugly bit of code which is faster than using `ETHER_ADDRESS_LEN CMOVE` with most VFX compilers. If your CPU does not support `W@` and `W!` as machine instructions, a simple `CMOVE` version may be faster.

`: [sm \ -- 0`
Starts the definition of a state machine's states.

`: smState \ n -- n+1`
Defines the next state as an `EQU` and increments the state number.

`: sm] \ n --`
Finishes the state machine and defines an equate of the number of states.

`: $>maca \ caddr len dest -- flag ; 0=good`
Convert an Ethernet MAC address string into six bytes at *dest*. If the string cannot be converted, *flag* is returned non-zero. The string *caddr/len* is in the form:

```
aa-bb-cc-dd-ee-ff
```

where the number base is hexadecimal.

14 Socket Primitives

The file SOCKETS\SOCKPRIM.FTH contains primitives for the sockets layers.

```
: GetSocketError        \ -- ErrorCode
```
Return the last socket error code.

```
: SetSocketError        \ ErrorCode --
```
Set the last socket error code.

```
: ClrSocketError        \ --
```
Clear the last socket error code.

```
MAXSOCKETS cells equ SOCKLIST_LEN       \ -- size
```
The size of the sockets array.

```
SOCKLIST_LEN buffer: socket_list        \ -- addr
```
The static sockets array.

```
  equ socket_list_last  \ -- addr
```
The address of the last entry in the socket list.

```
: initsockets   \ --
```
Initialise the socket list.

```
0 value NextSock#       \ -- socknum
```
Returns the next socket number to be used. This value is is ranged 0..n-1, not 1..n.

```
: +Sock#        \ --
```
Step to the next socket number, wrapping as required.

```
: NextSockEntry \ -- addr
```
Return the socket list entry address for `NextSock#`.

```
: add_hsocket   \ *socket -- HSOCKET|0
```
See if we can get a socket handle.

```
: sub_hsocket   \ *socket --
```
If the given socket address is in the table, mark its socket as unused. If the socket has already been removed, no action is taken.

```
: get_socket_addr       \ HSOCKET -- addr|0
```
Given a socket handle, return the socket address, or 0 if the socket is not in use.

```
: get_socket_inq        \ HSOCKET -- *q|0 ;
```
Return the socket's input queue.

```
: get_socket_outq       \ HSOCKET -- *q|0 ;
```
Return the socket's output queue.

```
: find_hsocket_port     { ip pcol port | hs *sk -- hsocket|0 }
```
Find a socket handle with matching IP, protocol and port.

```
: (find_hsocket_tcpport)        { ips ps ipd pd | hs *sk -- hsocket|0 }
```
Find a socket handle with matching IP and protocol and port

```
: find_hsocket_tcpport  { ips ps ipd pd | hs *sk -- hsocket|0 }
```
Find a socket handle with matching IP and protocol and port.

```
: WaitForIpAddress      \ --
```

Waits until we have a valid IP Address, i.e. one that is not 0.0.0.0.

`: numsockets { | res -- numsockets }`
Count the number of socket handles in use.

`: RebindAllSockets { ip -- }`
Rebind sockets to new local IP address.

14.1 Ephemeral ports

Ephemeral ports are port numbers allocated by TCP/IP when you make a connection. These are in a range which does not conflict with the "well known" ports. Each socket connection uses a different port number which allows TCP to identify multiple connections from one machine to the same port on another machine.

`#6000 equ FirstPortNumber \ -- n`
First ephemeral port number. Moved to *PNconfig.fth*.

`#9000 equ LastPortNumber \ -- n`
Last+1 ephemeral port number. Moved to *PNconfig.fth*.

`variable LastPortUsed \ -- addr`
Holds the last ephemeral port number used.

`: (GetFreePort) \ -- port`
Get a free port number; the free port numbers count up.

`: GetFreePort \ *socket -- port#`
Get an ephemeral port for a socket.

14.2 TCP control block creation and deletion

`: FlushQ \ queue --`
Flush the contents of a queue, return them to the free queue, and `FREE` the queue itself. If queue=0, no action is taken.

`: freeConnQ \ *cb --`
Free up any outstanding items on the connection queue.

`: DiscardTcpTcb \ *cb --`
Free-up all resources in the TCP control block, and then free the control block itself.

`: MakeTcpCb \ -- addr|0`
Create a new TCP control block from the heap. On error, zero is returned.

`: skTCP? \ *sk -- flag`
Returns true if the socket protocol is TCP.

`: FreeupSocket \ *sk --`
Frees up all allocated memory for a socket. **sk* is assumed to be valid.

`: ?BadSocket \ *sk x flag -- *sk x | -- 0 and exit caller`
A factor used by `SOCKET` for error recovery. If *flag* is non-zero, *x* is discarded, the socket is freed, and exit is from the calling word.

`: socket \ address-family socket-type proto-group -- hs|0`
Create a new socket of the given characteristics, returning the new socket number (1..n) on success, or zero for failure. **DO NOT CHANGE** the return of `INVALID_SOCKET`=zero for failure!

15 ICMP handling

The ICMP implementation is fairly minimal. Only the implemented functions are documented. Many diagnostics can be enabled by setting the equate `ICMPMON?` to non-zero.

See *ICMP.FTH*.

```
: ICMPcksum     \ *pb -- flag ; 0 for a pass
```
Generate the ICMP checksum. When used to check an ICMP header, it will return 0 for a good checksum.

```
: EchoRequest           \ *pbuf *icmp -- ; got a ping
```
Handle a PING request.

```
: RxICMPPacket          \ *pbuf -- ; OK so we got an IP packet
```
Handle an incoming ICMP packet.

16 Routing packets

The routing table is statically allocated. Configuration data is in *PNconfig.fth*.

```
#32 constant MAX_IPADDRS        \ -- n
```
No of entries in the routing table (not dynamic yet).

```
iproute_struct MAX_IPADDRS * equ RT_LEN \ -- n
```
Size of the routing table.

```
RT_LEN Buffer: IpRoutes         \ -- addr
```
This is the main routing table.

```
#7200000 constant ROUTE_LIFE    \ -- ms
```
Two hours in milliseconds.

```
#1000 constant ROUTE_SAMPLE_MS  \ -- ms
```
How often routes are tested im milliseconds.

```
variable RouteTicker    \ -- addr
```
Holds the time to sample route expiry next.

```
create EnetIPMask       \ -- addr
```
IP mask for addresses on Ethernet port. The data is in network order.

```
create IPGateway        \ -- addr
```
Gateway attached to Ethernet port. The data is in network order.

```
STRUCT iproute_struct   \ -- size ; of structure
```
Routing table entry. The fields have the prefix "IPROUTE_".

```
create defRoute \ --
```
IPROUTE structure for the default route.

```
: SetRouteLife  \ ms route --
```
Set the route's life to end *ms* from now.

```
: SetInfLife    \ route --
```
Set the route to have an infinite life.

```
: ExtendLife    \ ms route --
```
Extend the life of the given route by *ms* if the life is not infinite.

```
: InitRoutes    \ --
```
Initialise routing table.

```
: (find_route)  \ ip -- addr|0
```
Get routing table entry address (if available).

```
: AskForRoute   \ ip -- addr|0
```
See if we can make a route.

```
: ViaGateway?   { ip | iptmp *rt -- addr|0 }
```
Returns gateway route or none.

```
: CanWeRoute    { ip -- addr|0 ; returns route or 0 }
```
Handles routing via the subnet mask or gateway (if one exists).

```
: find_route    \ ip -- addr|0
```
Get table entry address (if available)

`: empty_route? \ -- addr|0`
Get table entry address (if available).

`: add_route \ ip -- addr|0`
Get table entry address (if available).

`: expire_routes? \ --`
Decrement the route entry timers and expire any routes that have timed out.

17 Basic IP layer

17.1 Tools

`: isIPforme \ *pbuf -- flag`
Flag true if this packet is for me.

`variable ipid \ -- addr ; for outgoing packets`
IP packet identifier, incremented by one for each outgoing IP packet.

`: SetIPhdr \ len ipsrc ipdest ipproto *iphdr --`
Initialise and fill in an IP header with no options.

`: pb>IPH \ *ip *pb --`
Step the PBuf data pointers back to the IP header from the TCP or UDP header.

17.2 Sending IP packets

`defer send>other \ *pb *rt -- len`
This allows other routes to be patched in to the stack without changing the stack source.

`: NoOther \ *pb *rt -- len`
The default action of `SEND>OTHER`.

`: (IPSend) \ *pb -- len|-1`
Send the data in the supplied PBuf (chain). *PB is the first pbuf in the chain (only 1 allowed) Returns the number of bytes sent if OK else `SOCKET_ERROR`.

`: IPSend \ *sk *pb -- len|SOCKET_ERROR`
Send the data in the supplied PBuf (chain). Returns number of bytes sent if OK else SOCKET_ERROR.

17.3 Receiving IP packets

`defer RouteIP \ *pb --`
This IP packet is not for me! Vectored routing allows later application dependent patching up. The default action is to discard the packet with `FreeQB`.

`defer CanRouteIP? \ *pb -- T|F`
Returns true if we can route this packet. Vectored routing allows later application dependent patching up.

`: NoCanRoute \ *pb -- False`
The default action of CANROUTEIP?

`: ProcessPacket \ *pb --`
Process an incoming packet

`Task PacketTask \ -- addr`
The task that handles incoming packets.

`: DoIncoming \ --`
The action of the incoming packet task.

`: RunIncoming \ --`
Start the incoming packet task.

`: GoodIPPacket { *pb -- }`
Process a good packet.

`: GoodIPPacket  \ *pb --`
Process a good packet.

`: RunIncoming   \ --`
Start the incoming packet task.

`: BadIpPacket   \ *pbuf --`
A header check failed, so discard the packet.

`: IPHdrLen      \ *ip -- len`
Find the length of the IP header.

`: +IPhdr        \ *ip -- addr`
Step from the IP header to the next item, e.g. TCP header.

`: RxIpPacket    \ *pb --`
Process a received IP packet.

18 ARP handler

ARP.fth handles ARP packets for Ethernet and DHCP.

```
$E1 equ ETHER_PORT      \ -- n
```
Port identifiers for Ethernet devices are in the range $E1..EF

```
create EtherBCastAddress        \ -- addr
```
Holds the Ethernet broadcast address FF-FF-FF-FF-FF-FF.

```
create EtherUnkAddress  \ --
```
Holds the Ethernet "unknown" address, all zeros.

```
: TurnEtherPacketRound  \ *eth_hdr --
```
Return the packet to the sender.

```
: HandleArpRequest      \ *pb ad *eth *arp --
```
Returns reply to an incoming ARP request.

```
: AddEtherRoute \ *etheraddr ipaddr --
```
Add an Ethernet route given a pointer to a 6 byte Ethernet address and an IP address.

```
: HandleArpReply        \ *pb *arp --
```
Handle an incoming ARP reply - so extract info.

```
: ArpForMe?     \ *arp -- flag ; nz=for me
```
Given the address of an incoming ARP payload, return true if the packet should be handled.

```
: EtherArpPacket        { *pb | ad ln *eth *arp -- }
```
Process an incoming ARP packet. The pbuf holds the received Ethernet packet. We use the incoming packet buffer for the reply.

```
create ProtoArpHdr      \ -- addr
```
Prototype ARP request header.

```
: SendArpRequest        \ ipaddr --
```
Formats and sends out an ARP request packet. This is always a broadcast.

```
: SendGratuitousArp     \ --
```
Send a "Gratuitous ARP Request".

19 UDP layer

```
  equ >UdpDataPos       \ -- offset
```
The UDP data offset in an outgoing UDP packet.

```
: UDPcksum      \ *hdr len ipsrc ipdest -- cksum
```
A generic UDP checksum function. The parameters supplied are used to form a 'pseudoheader' from which the acual checksum is generated.

```
: UDPSend       { hs *pb | *ip *sk -- len|err }
```
Send data on a connected UDP socket. *HS* is a handle to a UDP socket handle. **PB* is the address of a PBuff pointer already filled with the data to send (as UDP data). Returns the number of bytes sent if OK else SOCKET_ERROR.

```
: SendDatagram  { *pb ipaddr | *ip *udp -- }
```
Send data as a UDP packet without using a socket. The parameters are **pb*, a pointer to a PBuff containing the UDP data, and *ipaddr*, the IP address to send to.

```
: GetUDPAddress \ *pbuf -- *udp
```
Extract start of UDP data from IP packet.

```
: GetUDPLen     \ *udp -- len
```
Extract length of UDP packet.

```
: >UDPData      \ *udp -- *udpdata
```
Extract pointer to UDP packet data.

```
variable UDPServiceChain        \ -- addr
```
This variable anchors the chain that will be walked to find out what to do with a UDP packet that has arrived.

```
struct /UDPService      \ -- len
```
Describes the UDP service and how to run the service.

```
  int us.next           \ Link to next service in chain
  int us.port           \ UDP Port that this service handles
  int us.xtReceive      \ Word that will process this packet
  int us.xtIdle         \ Word that will run idle action
end-struct
```

```
: UDPprocessRx  \ *pb *udp --
```
Walk the UDP service chain and execute the first receive action for the port. If no association is found, try to route the packet.

```
: RxUDPPacket   { *pb | *udp hs *q bf -- }
```
Process a received UDP packet.

20 DHCP and BOOTP

The code in <*PNet*>*DHCP.fth* implements a DHCP **client** according to RFC2131. See also RFC2132 and RFC1542. Do not attempt to modify this code until you have absorbed these RFCs.

If you do not have a DHCP server on you network, there are many available. For Windows PCs, several users have recommended the **haneWin DHCP Server** from *www.hanewin.net*.

```
0 equ DHCPdebug?         \ -- x
```

Set this non-zero to display DHCP debug information during execution.

20.1 DHCP/BOOTP state machine

The DHCP system is controlled by a state machine, which is run whenever the timer *\fo{DHCPtimer) times out. The action performed is responsible setting the next state and updating the timer. The state machine actions are performed by `DHCPidle ( -- )` which is called in the service task.

States of the DHCP state machine. See RFC2131 Figure 5 below for the transition diagram.

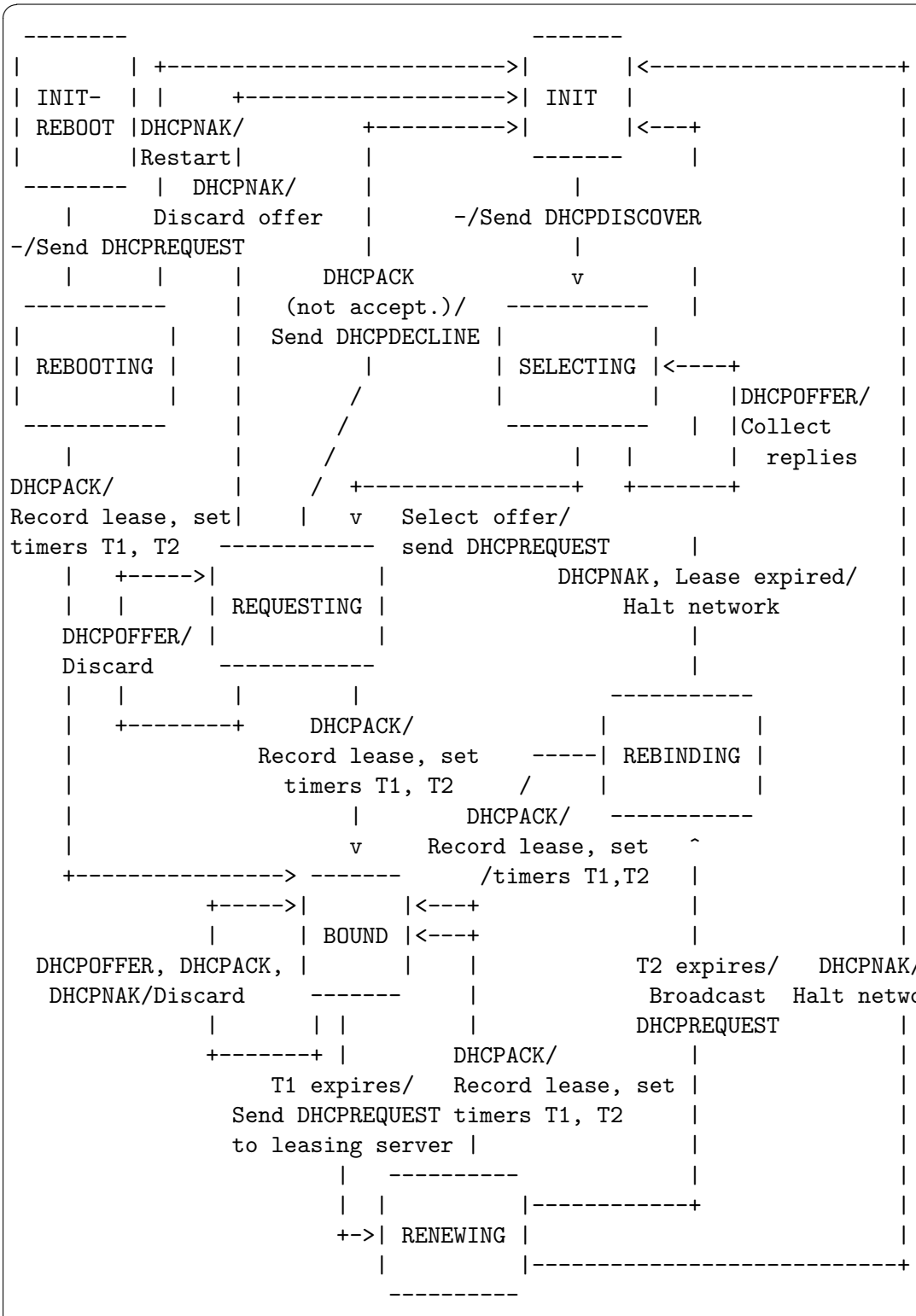


Figure 5: State-transition diagram for DHCP clients

```
[sm
  smState smDHidle      \ idle, do nothing (pseudostate)
  smState smDHinit      \ initialisation, o/p DHCPDISCOVER
  smState smDHselect    \ process DHCPOFFER(s)
  smState smDHrequest   \ process ACK, discard offers
                        \ ACK/decline -> o/p DHCPDECLINE, smDHinit
                        \ ACK/accept -> record lease, smDHbound
  smState smDHarpcheck  \ go ACK, check -> DHCPdecline/init or bound
```

```
  smState smDHbound     \ discard messages, time out ->
  smState smDHrenew     \ wait ACK, T2 expiry
  smState smDHrebind    \ wait ACK/NAK
  smState smDHrebooting \ warm restart, DHCPREQUEST, wait ACK/NAK
sm] #smDHactions        \ number of states in this machine
```

`variable DHCPTimer` `\ -- addr`
Holds the `TICKS` time at which the next DHCP action will be performed, 0 for no action.

`variable smDHcount` `\ -- addr`
Down counter used to provide repetitions and timeouts.

`variable smDHstate` `\ -- n`
Holds current DHCP state.

`create smDHactions` `\ -- addr`
Execution table containing the xts of the action words.

`: setDHstate` `\ xt state --`
`INTERPRETER` word that sets the action of the given state.

`: DHCPidle` `\ --`
Performed periodically to do DHCP actions and lease checks.

20.2 State machine utilitiles

`: SetDHCPtimer` `\ ms --`
Set the DHCP timer to time out *ms* later.

`: .DHstate` `\ --`
Debug tool to display the current state of the DHCP state machine.

`: toDHCPstate` `\ ms n state --`
The given *state* will be executed after the given period in *ms* and *n* times if sensitive to the number of times. Note that this specifies the interval until the state is executed, not the repetition rate of the state.

20.3 DHCP Data definitions

20.3.1 DHCP control data

These variables contain the working data used to access the DHCP server.

`variable BOOTP_XID` `\ -- addr`
Transaction ID number.

`variable DHCPLease` `\ -- addr`
Length of DHCP lease in ms.

`variable DHCPLeaseTimer` `\ -- addr`
`TICKS` at which lease times out.

`variable DHCP_T1` `\ -- addr`
Time to RENEW in ms.

`variable DHCP_T2` `\ -- addr`
time to REBIND in ms.

`variable DHCPServer \ -- addr`
DHCP server IP address.

`wvariable DHCPflags \ -- addr`
Holds the DHCP flags, set to $8000 to force broadcast responses, or 0 when we can accept unicast responses.

20.3.2 DHCP transient data

These variables contain data generated by the DHCP transactions. It is later manipulated to set system variables and the DHCP control data after validation.

`variable GivenIP \ --`
IP address given by DHCP or BOOTP.

`variable GivenMask \ --`
IP mask given by DHCP or BOOTP.

`variable GivenGateway \ --`
IP gateway given by DHCP or BOOTP.

`variable GivenServer \ --`
IP address of DHCP server given by DHCP or BOOTP.

`variable GivenLease \ -- addr`
Length of DHCP lease in seconds given by DHCP or BOOTP.

`variable GivenT1 \ -- addr`
Time to RENEW in ms given by DHCP or BOOTP.

`variable GivenT2 \ -- addr`
time to REBIND in ms given by DHCP or BOOTP.

`variable GivenReply? \ -- addr`
Set non-zero when a DHCP reply has been received, but the data has not yet been been processed. During this time any following DHCP packets will be discarded.

`variable GivenSNTP \ -- addr`
SNTP server returned by DHCP.

`variable GivenDNS \ -- addr`
DNS server returned by DHCP.

20.3.3 DHCP packet layout

`STRUCT /dhcp \ -- len`
BOOTP and DHCP payload structure after the IP and UDP headers.

`CREATE BOOTPMAGIC \ -- addr`
Magic number used to identify BOOTP and DHCP packets.

20.4 DHCP tools

`: umin \ u1 u2 -- u1|u2`
Minimum of two unsigned values.

`: TxMs \ secs -- ms`
Return the number of seconds required for expiry of the T1/T2 timer.

`: NoDHCPserver \ --`

Set up for no known DHCP server.

`: AcceptDHCPin \ --`
Accept incoming DHCP packets by clearing `GivenReply?`.

`: RejectDHCPin \ --`
Reject incoming DHCP packets by setting `GivenReply?` to an non-valid value.

`: AcceptGiven \ --`
Transfer the data given by DHCP transactions to the working data.

20.5 DHCP state selection

Each state is controlled by three values defined here.

- Entry delay - delay before state is executed for the first time.
- Period - period between repetitions of the same state.
- Count - number of repetitions before `smDHcount` reaches zero.

`: goIdle \ --`
Go to the `smDHidle` state.

`: goInit \ --`
Go to the `smDHnit` state, so starting the DHCP process.

`: goSelect \ --`
Go to the `smDHselect` state.

`: goRequest \ --`
Go to the `smDHrequest` state.

`: goARPcheck \ --`
Go to the `smDHselect` state.

`: goBound \ --`
Go to the `smDHbound` state.

`: goRenew \ --`
Go to the `smDHrenew` state.

`: goRebind \ --`
Go to the `smDHrebind` state.

`: goRebooting \ --`
Go to the `smDHreboot` state.

20.6 UDP transmission

`: (BOOTPSend) { *pb ipaddr -- }`
Send BOOTP data as a UDP packet. The parameters are **pb*, a pointer to a PBuff containing the UDP data, and *ipaddr*, the IP address to send to.

20.7 Outgoing message tools

Some data is carried as DHCP options in the DHCP packet. Options that may contain an odd number of bytes are forced to a 16 bit boundary.

`: c!++ \ addr b -- addr+1`

`COMPILER` word, saves *b* at *addr* and increments *addr*.

`: !(n)++ \ addr x -- addr+4`
`COMPILER` word, saves *x* at *addr* and increments *addr*.

`: haligned \ *opt -- *opt'`
Align option - assumes that options start 16 bit aligned. If necessary a zero (pad) byte is added.

`: INCLUDE_SERVERID_OPT \ *opt -- *opt'`
Add served IP address to the options list.

`: INCLUDE_REQ_INF_LEASE_OPT \ *opt -- *opt'`
Add request for an ininite lease to the options list.

`: INCLUDE_OFFERED_LEASE_OPT \ *opt -- *opt'`
Add the offered lease to the options list.

`: INCLUDE_REQUESTADDR_OPT \ *opt -- *opt'`
Add the offered IP address to the options list.

`CREATE ParamRequestList`
Parameter Request List that indicates which responses we want.

`: INCLUDE_PARAMREQ_OPT \ *opt -- *opt'`
Add parameter request list to the options list.

`: addDHCPopts \ msgtype *bootp --`
Add the relevant options for the message type to the options section of the DHCP message.

`: UseBroadcast? \ ServerAddr IPAddr -- ServerAddr`
If ServerAddr and IPAddr are not the same and IPAddr is not INADDR_BROADCAST (the initialised value) then substitute INADDR_BROADCAST for ServerAddr. So if siaddr is left at 0 in the DHCP offer the Request will be broadcast.

`: GetBOOTPServerAddr \ -- ipaddr`
If there is any doubt about the server IP address then broadcast the reply. At this point the routing table only contains entries for the received BOOTP reply and the broadcast address so there is no point in replying to the siaddr or the address in DHCPServerID if it differs from BOOTPServer.

`: .DHCPtype \ msgtype --`
Display the message type as text.

`: SendDHCP { msgtype | *pb *bootp -- }`
Send a DHCP message. The fields and options are filled in as appropriate for the message type.

`: SendDiscover \ --`
Send a DHCPDISCOVER message.

`: SendRequest \ --`
Send a DHCPREQUEST message.

`: SendDecline \ --`
Send a DHCPDECLINE message.

`: doDHidle \ --`
The idle action is just to restart the timer.

`: doDHinit \ --`
Send DHCPDISCOVER periodically until we get a response

```
: doDHselect    \ --
```
Select is currently a no-op as we take the first offer we receive.

```
: doDHrequest   \ --
```
Send DHCPREQUEST periodically until we get a response

```
: doDHarpcheck  \ --
```
The DHCP specification says that clients **SHOULD** check that the IP address given to them is not used elsewhere. If a route is found, the process is restarted.

```
: doDHbound     \ --
```
Wait until the T1 timer expires, and go to the RENEWING state.

```
: doDHrenew     \ --
```
Send DHCPREQUEST periodically until we get a response.

```
: doDHrebind    \ --
```
Send DHCPREQUEST periodically until we get a response.

```
: doDHrebooting \ --
```
Send DHCPREQUEST periodically until we get a response.

20.8 Receive BOOTP/DHCP packet

DHCP packet reception is treated as a special case because DHCP uses two ports and must operate before the unit's IP address has been set.

```
: doDHCPopt     \ *option -- *option'
```
Process the given option if we know about it and step over it. Unknown options are ignored. Values for known options are saved in the `Givenxxx` variables.

```
: scanDHCPopts  \ *bootp len --
```
Given the DHCP data area, parse the DHCP options.

```
: RxDHCP        { *pb | *udp *bootp len -- }
```
Process received DHCP and BOOTP packets. When we get here we know this is from port 67 to port 68 and that the Ethernet packet was either addressed to us or was broadcast.

```
: DHCPpacket?   \ *udp -- flag
```
Given a UDP header, return non-zero if the packet is a DHCP packet. Used in the UDP layer to route DHCP packets.

20.9 State machine initialisation and startup

```
  ' doDHidle       smDHidle      setDHstate
  ' doDHinit       smDHinit      setDHstate
  ' doDHselect     smDHselect    setDHstate
  ' doDHrequest    smDHrequest   setDHstate
  ' doDHarpcheck   smDHarpcheck  setDHstate
  ' doDHbound      smDHbound     setDHstate
  ' doDHrenew      smDHrenew     setDHstate
  ' doDHrebind     smDHrebind    setDHstate
  ' doDHrebooting  smDHrebooting setDHstate


#68 equ DHCPport        \ -- port
```
DHCP standard port.

`create DHCPServiceStruct \ -- addr`
Holds the UDP Service info to run the DHCP code on receipt of DHCP packets, or idle on timeout. Note that this **must** match the `/UDPservice` structure in *UDP.fth*.

`: runDHCP \ --`
Start the DHCP process and wait until a valid IP address has been set.

21 DNS client

The code in `DNS.fth` implements a simple DNS client. The focus of the design is on minimising overall RAM usage. All data is transferred using UDP.

The code is compiled after the BSD layer because it makes heavy use of that layer.

The main documentation for DNS is contained in RFCs 1034 and 1035. The language is abstruse at best.

21.1 Configuration

These equates and data are only used if not already defined.

```
1 equ DNSdebug?          \ -- x
```

Set this non-zero to display DNS debug information during execution.

```
#53 equ DNSport#         \ -- port#
```

The standard DNS port number on the server.

```
3 equ #DNS               \ -- u
```

Maximum number of DNS attempts if no response.

```
5000 equ /DNSms          \ -- ms
```

Maximum time (in milliseconds) allowed for a DNS transaction.

```
#512 equ /DNS            \ -- u
```

Maximum size of the DNS payload.

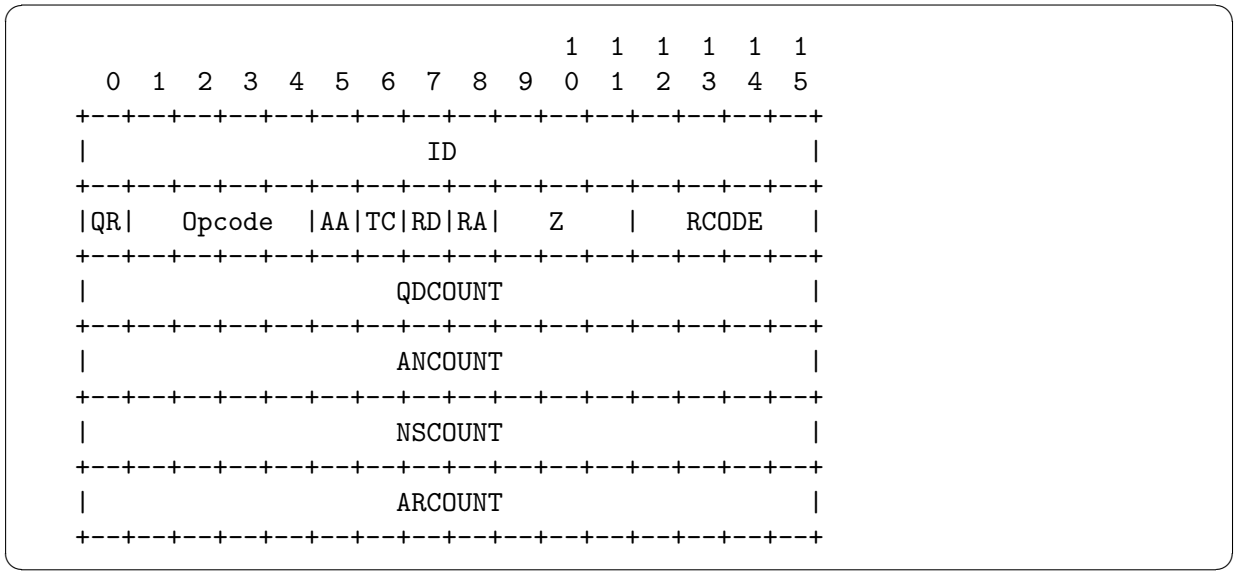
```
variable DNSserver       \ -- addr
```

Holds DNS server or 0 if not configured yet.

21.2 Queries and responses

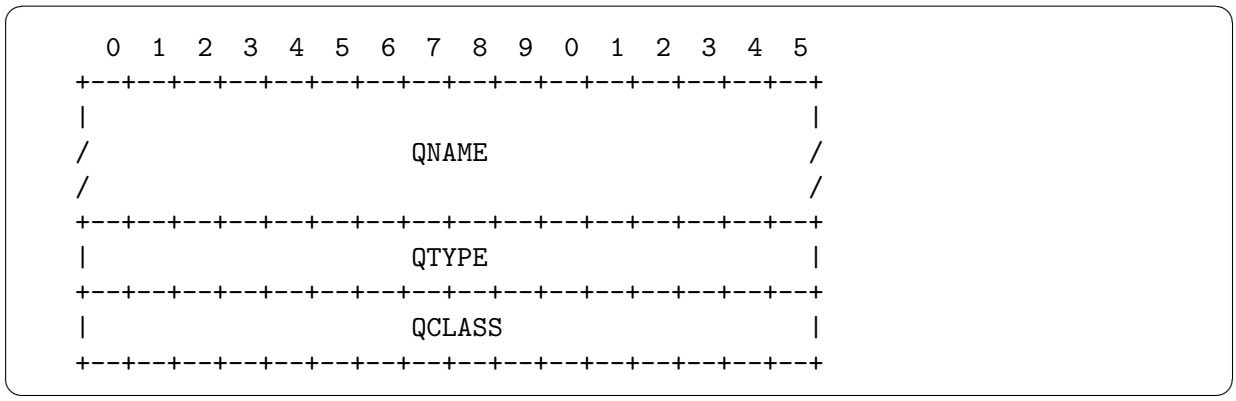
For the full details of DNS queries and responses, see RFCs 1034 and 1035. What follows in this documentation is a small subset.

The header (RFC 1035, 4.1.1) contains the following fields:



A query section contains the following fields, QNAME (RFC 1035, 4.1.2), QTYPE (RFC 1035, 3.2.2) and QCLASS (RFC 1035, 3.2.4/5).

The question section is used to carry the "question" in most queries, i.e., the parameters that define what is being asked. The section contains QDCOUNT (usually 1) entries, each of the following format:



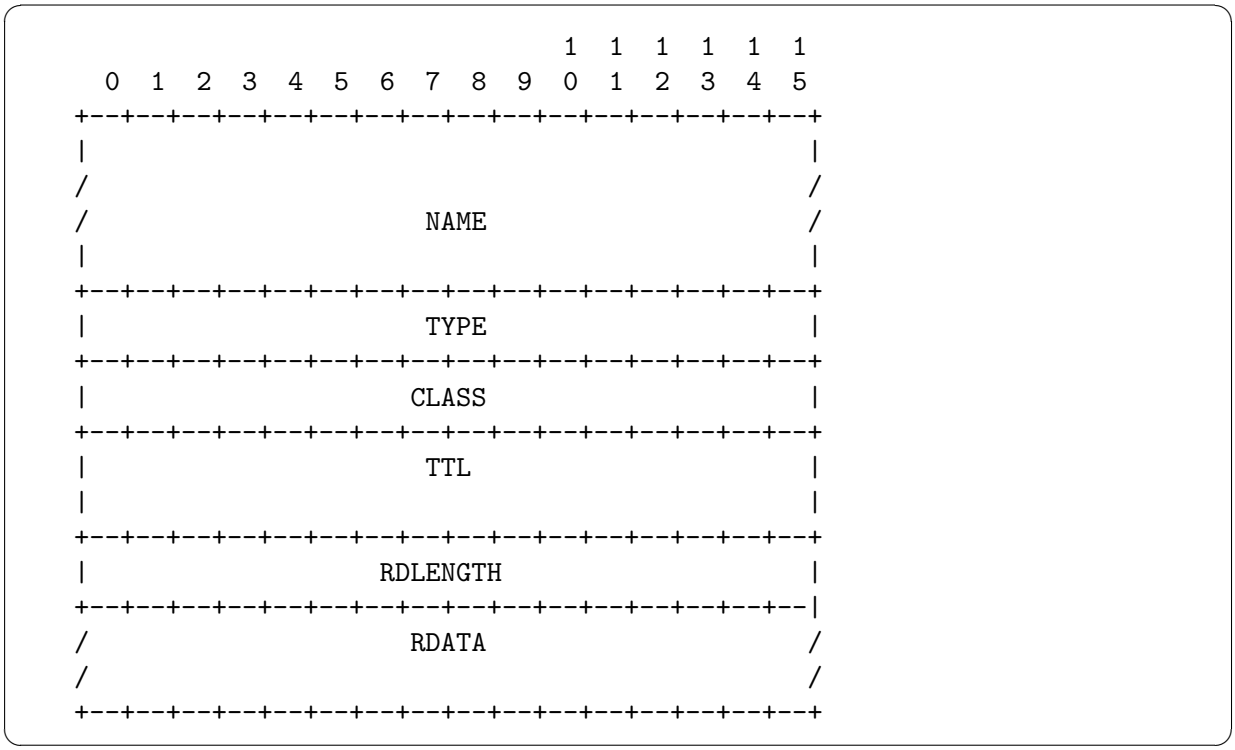
where:

`QNAME` a domain name represented as a sequence of labels, where each label consists of a length octet followed by that number of octets. The domain name terminates with the zero length octet for the null label of the root. Note that this field may be an odd number of octets; no padding is used.

`QTYPE` a two octet code which specifies the type of the query. The values for this field include all codes valid for a TYPE field, together with some more general codes which can match more than one type of RR.

`QCLASS` a two octet code that specifies the class of the query. For example, the QCLASS field is IN for the Internet.

The answer, authority, and additional sections all share the same format: a variable number of resource records, where the number of records is specified in the corresponding count field in the header. Each resource record has the following format:



where:

`NAME` a domain name to which this resource record pertains.

`TYPE` two octets containing one of the RR type codes. This field specifies the meaning of the data in the RDATA field.

`CLASS` two octets which specify the class of the data in the RDATA field.

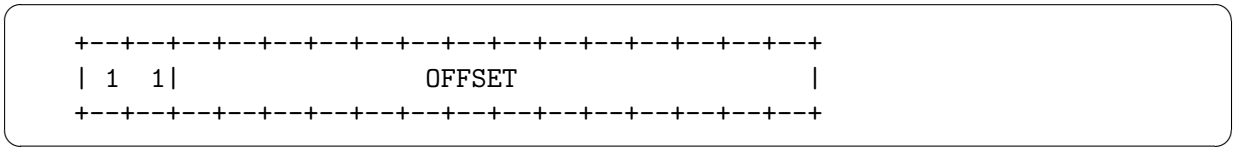
`TTL` a 32 bit unsigned integer that specifies the time interval (in seconds) that the resource record may be cached before it should be discarded. Zero values are interpreted to mean that the RR can only be used for the transaction in progress, and should not be cached.

`RDLENGTH` an unsigned 16 bit integer that specifies the length in octets of the RDATA field.

`RDATA` a variable length string of octets that describes the resource. The format of this information varies according to the TYPE and CLASS of the resource record. For example, the if the TYPE is A and the CLASS is IN, the RDATA field is a 4 octet ARPA Internet address.

In order to reduce the size of messages, the domain system utilizes a compression scheme which eliminates the repetition of domain names in a message. In this scheme, an entire domain name or a list of labels at the end of a domain name is replaced with a pointer to a prior occurance of the same name.

The pointer takes the form of a two octet sequence:



The first two bits are ones. This allows a pointer to be distinguished from a label, since the

label must begin with two zero bits because labels are restricted to 63 octets or less. (The 10 and 01 combinations are reserved for future use.) The OFFSET field specifies an offset from the start of the message (i.e., the first octet of the ID field in the domain header). A zero offset specifies the first byte of the ID field, etc.

21.3 Tools

`4 buffer: DNSid \ -- addr`
Holds the DNS transaction number. The number is incremented before use so that a good response should have the same number.

`create DNSQtemplate \ -- addr`
Template for a DNS query.

`#12 equ /DNSQtemplate \ -- len`
Length of the DNS query template.

`: genDNSQhead \ addr -- addr`
Generate a DNS query header at *addr* and return the address just after the header.

`: c!++ \ addr b -- addr+1`
Saves *b* at *addr* and increments *addr*.

`: genLabel \ caddr len addr -- addr'`
Add a label (n + n bytes) to the text at addr, returning the next label address.

`: genDNSname \ caddr len addr -- addr'`
Generate a name as a sequence of labels.

`: genDNSquestion \ caddr len addr -- addr'`
Generate the question section of a query.

`: skipDNSQ \ caddr len -- caddr' len'`
Skip the query portion of a DNS reponse.

`: skipRRname \ caddr len -- caddr' len'`
Step over the name field in a resource record (RR).

`: cleanDNSq \ hs buffer --`
Clean up the query by releasing the buffer and closing the socket.

21.4 User words

After acquisition of the DNS server's IP address, either by specification or by DHCP, the only function provided is to convert a domain name string, e.g. *www.google.com* into an IPv4 address.

`: DNSquery \ caddr len -- ip|0`
Send a UDP DNS query message requesting the IP address of the given domain name. Returns the IP address for success, or zero on failure.

22 SNTP client

As of PowerNet v4.30 (8 May 2008) the user interface to the SNTP data has changed and is not compatible with the previous one.

The code in <*PNet*>*SNTP* implements most of an SNTP **client** according to RFC1361 and RFC4330. See also RFC1305. Do not attempt to modify this code until you have absorbed these RFCs.

If you do not have an NTP server on your network, there are "public" ones available. See *www.ntp.org* for details. Note when testing that advertised public servers are not always available. Your ISP probably maintains a functioning NTP server.

```
0 equ SNTPdebug?        \ -- x
```
Set this non-zero to display SNTP debug information during execution.

```
1 equ SNTPactions?      \ -- x
```
Defined the actions performed by default for timestamps.

```
0     Dummy action
1     Set clock
...   user defined action
```

22.1 SNTP equates and structure

```
#123 equ SNTPport#      \ -- port#
```
The standard SNTP port number. The client (us) and the server use the same port number.

```
STRUCT /sntp    \ -- len
```
SNTP payload structure after the IP and UDP headers. The *Timestamp* fields contain a 32.32-bit fractional time in network (big-endian) format. The first cell contains seconds, measured since 1st Jan 1900, as per RFC1361. The second cell contains a fractional second.
If you are using the SNTP code with the Unix calendar code in *Examples/UnixTime.fth*\, NTP timestamps are based from 1 Jan 1900, which has an `LSECONDS` value of 1291876096+86400=1291962496, so add 1291962496 to the NTP seconds value to produce a Unix time value.

If you are using this code with NTP timestamps, these are based from 1 Jan 1900, which has an `LSECONDS` value of 1291876096+86400=1291962496, so add 1291962496 to the NTP seconds value to produce a Unix time value. Note that the Unix `LSECONDS` counter rolls over in 2036. Suitable code to convert `LSECONDS` to time and date can be found in your cross compiler *Examples/UnixTime.fth*.

If you are planning a product which needs to cope with dates beyond 2035, note the following from RFC4330.

As the NTP timestamp format has been in use for over 20 years, it is possible that it will be in use 32 years from now, when the seconds field overflows. As it is probably inappropriate to archive NTP timestamps before bit 0 was set in 1968, a convenient way to extend the useful life of NTP timestamps is the following convention: If bit 0 is set, the UTC time is in the range 1968- 2036, and UTC time is reckoned from 0h 0m 0s UTC on 1 January 1900. If bit 0 is not set, the time is in the range 2036-2104 and UTC time is reckoned from 6h 28m 16s UTC on 7

February 2036. Note that when calculating the correspondence, 2000 is a leap year, and leap seconds are not included in the reckoning.

22.2 SNTP Configuration

`struct /SNTPparams \ -- len`

A set of parameters to configure the SNTP service This structure is the simplest way to interact with the SNTP module. The elements have the following meaings :-

`sp.SNTPserver`
- the IPv4 address of the SNTP server to use.

`sp.PollInterval`
- log2 of the interval between polls in seconds, e.g. 5 for a 32 second interval. This value is set to 5 by default, and you should probably increase it when you are satisfied with your SNTP handling.

`*\fo{sp.xtSetTime`
- the xt of a callback to your application to process the received SNTP packet. This is called whenever an SNTP packet is received. The word receives the address and length of the UDP data, i.e. a `/SNTP` structure described below, and should return nothing `( caddr len -- )`. Note that the data is in a `pbuff` that is owned by the SNTP engine and will be discarded. Copy required data out of the `/SNTP` structure.

`sp.xtGetTime`
- an optional configuration, and is not required for pure SNTP. If set, it must put the current timestamp on the stack as a 32.32-bit fractional RFC1361 time. The stack effect is `( -- TimeLo TimeHi )`. The default action is a word that returns a zero timestamp, which is SNTP-compliant behaviour. This configuration option is supplied to give the system better resolution if the clock controller can generate an NTP-compliant timestamp.

```
  int sp.SNTPserver     \ IP address of the server to use. Must be first.
  int sp.PollInterval   \ Time between SNTP polls, in seconds, as a power of two
  int sp.xtSetTime      \ Callback to alert caller to a new NTP packet
  int sp.xtGetTime      \ xt of word to provide current timestamp, or zero if n/a
end-struct
```

`idata create SNTPparams \ -- addr`

The `/SNTPparams` parameter block in IDATA space (RAM). Note: This layout **must** match the layout of the `/SNTPparams` structure above. Patch this table with the actions you supply.

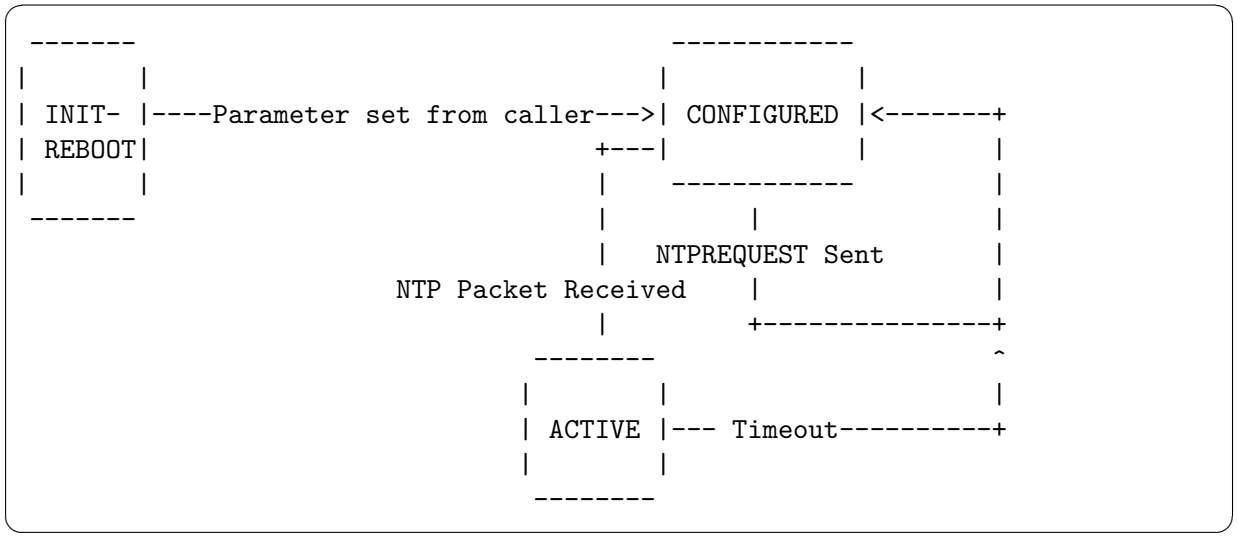
`SNTPparams equ SNTPserver`

Address holding SNTP server IP address.

22.3 SNTP state machine

The SNTP system is controlled by a state machine, which is run whenever the timer *\fo{SNTPtimer) times out. The action performed is responsible for setting the next state and updating the timer. The state machine actions are performed by `SNTPidle ( -- )` which is called in the service task.

States of the SNTP state machine. See RFC1361




```
[sm
  smState smSNinit              \ initialisation, wait for params from caller
  smState smSNconfigured        \ Send SNTP Request, set up timeout
  smState smSNactive            \ SNTP answer received within timeout period
sm] #smSNactions                \ number of states in this machine
```

`variable SNTPTimer      \ -- addr`
Holds the `TICKS` time at which the next SNTP action will be performed, 0 for no action.

`variable smSNcount      \ -- addr`
Down counter used to provide repetitions and timeouts.

`variable smSNstate      \ -- n`
Holds current SNTP state.

`create smSNactions      \ -- addr`
Execution table containing the xts of the action words. This in `CDATA` space and is filled in later.

`: setSNstate    \ xt state --`
`INTERPRETER` word that sets the action of the given state.

`: SNTPidle      \ --`
Performed periodically to do SNTP actions.

22.4 State machine utilities

`: SetSNTPtimer  \ ms --`
Set the SNTP timer to time out *ms* later.

`: .SNstate      \ --`
Debug tool to display the current state of the SNTP state machine.

`: toSNTPstate   \ ms n state --`
The given *state* will be executed after the given period in *ms* and *n* times if sensitive to the number of times. Note that this specifies the interval until the state is executed, not the repetition rate of the state.

22.5 Outgoing message tools

`variable SNTPreply?     \ -- addr`
Set non-zero when an SNTP reply has been received, but the data has not yet been been processed. During this time any following SNTP packets will be discarded.

`: (SNTPPSend)  { *pb ipaddr -- }`
Send SNTP data as a UDP packet. The parameters are **pb*, a pointer to a PBuff containing the UDP data, and *ipaddr*, the IP address to send to.

`: SendSNTP  { | *pb *sntp -- }`
Send an SNTP message.

22.6 SNTP state selection

Each state is controlled by three values defined here.

- Entry delay - delay before state is executed for the first time.
- Period - period between repetitions of the same state.
- Count - number of repetitions before `smSNcount` reaches zero.

`: goSNTPInit  \ --`
Go to the `smSNnit` state - stop SNTP activity and wait for parameters from the calling app.

`: goSNTPConfigured  \ --`
Go to the `smSNconfigured` state; start the SNTP process.

`: goSNTPActive  \ --`
Go to the `smSNactive` state - SNTP data has been received

`: doSNinit  \ --`
`SNTPauto?`=nz: Do nothing until SNTP server has been set, either manually or by DHCP, and then start SNTP.

`: doSNconfigured  \ --`
"Configured" state has timed out - send SNTP request and wait for response or timeout

`: doSNactive  \ --`
"active" state has timed out; revert to "configured"

`: runSNTP  \ --`
Start the SNTP process and wait until a valid time has been returned.

22.7 Receive SNTP packet

`: RxSNTP  \ *pb --`
Process received SNTP packets. When we get here we know this is from port 123 and that the Ethernet packet was either addressed to us or was broadcast.

`create SNTPServiceStruct  \ -- addr`
Holds the UDP Service info to run the SNTP code on receipt of SNTP packets, or idle on timeout. Not that this **must** match the `/UDPService` structure in *UDP.fth*.

22.8 Set up state machine

```
' doSNinit          smSNinit       setSNstate
' doSNconfigured    smSNconfigured setSNstate
' doSNactive        smSNactive     setSNstate
```

23 TCP layer

You will find the books TCP/IP Illustrated, Volume 1 (Stevens) and Volume 2 (Wright & Stevens) useful when exploring the PowerNet TCP code, especially for details of sequence and ack numbers.

23.1 TCP configuration

The configuration below can now be found in *PNconfig.fth*.

`0 equ TCPDEBUG  \ -- n`
Set this flag non-zero to generate debug messages.

`#1460 equ TCPDATASIZE   \ -- n ; EGS009`
Transmit buffer size of pbuf less tcp hdr(ish).

`#1460 2 * equ TCPTXBUFFSIZE     \ -- n`
Size of the retransmission buffer for a TCP socket. One buffer of this size is allocated from the heap when a TCP socket is created.

`#1460 equ TCPWINDOWSIZE \ -- n ; EGS002`
TCP window size.

`#100 equ TXDELAYTIME    \ -- ms`
timer for delayed transmit (mSecs), default is 100 milliseconds.

`#10 equ ACKDELAYTIME    \ -- ms`
timer for delayed ack (mSecs), default is 10 milliseconds.

`#5000 equ TXRETRYTIME   \ -- ms`
timer for transmit retries (mSecs), default is 5 seconds.

`#30000 equ TCPCONNECTTIME       \ -- ms`
timer for incoming connections to complete (mSecs), default is 30 seconds.

`#30000 equ TCPMSLTIME   \ -- ms`
timer for maximum segment lifetime (mSecs, default is 30 secs).

`#7200000 equ TCPIDLETIME        \ -- ms`
timer for idle disconnection (mSecs, default is 2 hours).

`#12 equ TCPMAXRETRIES   \ -- n`
Maximum number of transmission retries before returning SOCKET_ERROR, default is 12. If a receiving socket fails to accept new data, the total timeout will be TXRETRY-TIME*TCPMAXRETRIES, the default being 60 seconds.

`create MSSopt   \ -- addr`
Maximum Segment Size option.

23.2 Unknown socket requests

`variable NoTcpHandle    \ -- addr`
Holds the handle of a socket used to handle unknown TCP requests.

`: Init-NoTcp-Socket     \ -- res|SOCKET_ERROR`
Initialise a socket structure to handle unknown TCP requests. `SOCKET_ERROR` is returned if the socket cannot be initialised.

23.3 TCP structures and equates

Most equates are not documented.

`STRUCT tcp_hdr  \ -- size ; of tcp header structure`
TCP header structure.

`STRUCT tcpcb    \ -- size`
TCP control block structure is defined in *structs.fth*.

`: TcpState@     \ *sk -- state`
Get socket state.

`: .tcpstate     \ n --`
Display tcp socket state. Only compiled if DIAGS? is non-zero.

`: ShowTcpState  \ *sk --`
Display the state for the connection

23.4 TCP structure creation and deletion

`: initTcpTxBuff \ buff *cb --`
Use *buff* to initialise or reset the TCP transmit buffer.

`: -TcpTxBuff    \ *cb --`
`FREE` the TCP transmit buffer and reset the transmit pointers.

`: goCLOSED      \ *sk --`
Place socket in closed state.

`: (CloseSocket) \ *sk --`
Prepare a socket for closure. Final closure and memory recovery is performed in the TCP idle action to avoid race hazards.

`: ChkTcpTxBuff  \ *cb -- ior`
If the TCPCB does not have a transmit buffer, create it. Return 0 if the buffer exists or has been created. We assume that the TCPCB is valid.

23.5 TCP header use

`: GetTCPHdrLen  \ *tcp -- len`
Extract length of TCP header (varies with options).

`: GetTCPPktLen  \ *ip -- len ; SFP005`
Extract length of TCP packet

`: >TCPData      \ *tcp -- *tcpdata`
Get pointer to TCP packet data.

`: GetTCPDataLen \ *ip -- len ; SFP005`
Extract length of TCP data packet.

`: GetMSS        \ *tcp defmss -- mss ; SFP045`
If the TCP header in a SYN or SYN/ACK packet contains an MSS option return it, otherwise return the default value. the output is limited to the range `0..defMSS`.

`: GetMSSsyn     \ *tcp -- mss ; SFP045`
If the TCP header in a SYN packet contains an MSS option return it, otherwise return the default value of 536. Used by servers to read the incoming MSS in a SYN packet.

`: GetMSSsyn/ack \ *tcp -- mss ; SFP045`
If the TCP header in a SYN packet contains an MSS option return it, otherwise return the default value of 536. Used by clients to read the incoming MSS in a SYN/ACK packet.

23.6 TCP checksum handling

`: TcpCksum \ *tcp tcplen iprem iploc -- cksum`
A generic TCP checksum function. The parameters supplied are used to form a 'pseudoheader' from which the actual checksum is generated.

`: GenTcpCksum \ *sk *tcp len --`
Generate the TCP checksum using a prepared header and socket data for the IP addresses. Len is the length of the full TCP packet.

23.7 TCP window size

The words in this section are the defaults for transmitting the TCP receive window size to the remote end.

PowerNet queues input packets in the socket structure before passing them to the BSD layer. Usually, ACKs are delayed. There are three ways to implement receive window handling, selected by the equate `genWinSize?` in the PowerNet configuration file.

- `genWinSize?`=0. The receive window is always set to `TCPWINDOWSIZE`.
- `genWinSize?`=1. By default, we assume that if all the input packets have not been consumed by the time the ACK is sent, the system is under heavy load and we do not want any more input for the moment. This is indicated by setting the window size to zero. Otherwise, the receive window is set to `TCPWINDOWSIZE`. Although this "bang bang" approach is very crude, it works well unless many packets are being sent from PowerNet, as can happen when web pages are served from memory.
- `genWinSize?`=2. The word `genWindowSize ( *sk *tcp -- )` is provided by you to suit your application. See *tcp.fth* for the examples.

`: genWindowSize \ *sk *tcp --`
Given a socket **sk* and a TCP header at **tcp*, calculate the TCP window size and place it in the TCP header. This version is used when `genWinSize?`=0, and implements the previous constant window size.

`: skMoreIp? \ *sk -- flag`
Return true if the socket has an input packet waiting.

`: NextWinSize \ *sk -- size`
Return the TCP Window size to be used for the packet send.

`: genWindowSize \ *sk *tcp --`
Given a socket **sk* and a TCP header at **tcp*, calculate the TCP window size and place it in the TCP header. This version is used when `genWinSize?`=1, and implements the "bang bang" control mechanism.

23.8 TCP transmission primitives

`: #unsent \ *cb -- n`
Number of bytes waiting to be sent. If no transmit buffer has been allocated, this returns 0.

`: #unacked \ *cb -- n ; SBD011`

Returns the number of sent bytes awaiting acknowledgement. If no transmit buffer has been allocated, this returns 0.

`: setAckDelay  \ *cb --`
Set the ACK delay timer.

`: checkRxWin  \ *sk --`
If the receive window is set to 0 and the delayed ACK timer is not running, start the delayed ACK timer.

`: checkTxUnsent \ *cb --`
The TCP transmit primitive *\fo{SendTcpPkt) always clears the delayed ACK timer. If there is no more data to send, clear the delayed transmit flag and set the delayed ACK timer if if the receive window is zero.

`: MakeTCPhdr  \ fl ol *sk *tcp --`
Take data from the socket **sk* and create a TCP header at **tcp*. The TCP header checksum is set to 0.

`: SendTcpPkt { *sk fl *opt ol *dat dl | *pb *tcp pl -- res }`
Transmit a TCP packet directly through the IP layer.

`*sk`	points to the socket struct to be used
`fl`	the flag bits to use
`*opt,ol`	are the option pointer/length (if non-zero)
`*dat,`	dl are the data pointer/length (if non-zero)
`res`	= SOCKET_ERROR on failure, or no. of bytes sent on success

`: SendTcpOpts  \ *sk fl *opt ol -- ; SFP005`
Send a packet that consists of the flags and options without data.

`: SendTcpFlags  \ *sk fl -- ; SFP005`
Send a packet that just consists of the flags, without options or data.

`: ForceTCPReset \ *sk -- ; SFP012`
Force a socket reset, and then close the socket.

`: sent<txbuff  \ *cb len --`
Bump the transmit buffer after len bytes sent from txbuffer.

`: SendBuff>TCP  { *cb *sk | nb -- }`
Send any buffered data out. No action is taken if there is nothing to send. Use this only in the TCP state machine.

`: SendAck>TCP  { *cb *sk | nb -- }`
Send any buffered data out with an ACK.

`: skValid?  \ *sk -- flag`
Returns true if the socket structure is a valid TCP socket.

`: skCanSend?  \ *sk -- flag`
Returns true if the socket structure is valid, the socket is in TCPS_ESTABLISHED state and has not exceeded the maximum number of retry attempts.

`: sendtoTCP  { hsocket *bf len flags | bufflen *sk *cb nb -- res }`
Move user data into the socket's transmit buffer, waiting in the word until all data has been transferred to the buffer. This may involve packet transmission while waiting for the buffer

to empty. Because the socket may be closed during transmission, the socket must be checked before each buffer fill operation. Note that parts of this code must not be pre-empted by the scheduler.

23.9 TCP state primitives

```
: isTCP_Fin     \ *tcp -- T|F
```
Set if FIN flag bit is set.

```
: isTCP_Syn     \ *tcp -- T|F ; set if flag bit is set
```
Set if SYN flag bit is set.

```
: isTCP_Rst     \ *tcp -- T|F ; set if flag bit is set
```
Set if RST flag bit is set.

```
: isTCP_Psh     \ *tcp -- T|F ; set if flag bit is set
```
Set if PSH flag bit is set.

```
: isTCP_Ack     \ *tcp -- T|F ; set if flag bit is set
```
Set if ACK flag bit is set.

```
: isTCP_Urg     \ *tcp -- T|F ; set if flag bit is set
```
Set if URG flag bit is set.

```
: isTCP_SYN/ACK \ *tcp -- flag ; SFP005
```
Set if SYN and ACK flags are both set.

```
: SendNoSocket  { *pb *tcp | *ip *sk *cb -- }
```
Handles sending the necessary RST replies when a connection request is made for a TCP socket we don't know about. Go send an RST message. The packet buffer is freed.

```
: set2MSL       \ *cb --
```
Start the 2xMSL timer for disconnection.

```
: setIdle       \ *cb --
```
Start the disconnect timer with the IDLE period.

```
: setConn       \ *cb --
```
Start the connect timer with the TCPCONNECTTIME period.

```
: goSYN_REC     \ *sk --
```
Put the given socket into TCPS_SYN_RECEIVED state. This transition is performed by a server.

```
: goESTABLISHED \ *sk --
```
Move the socket into ESTABLISHED state.

```
: goLAST_ACK    \ *sk --
```
Move the socket into LAST_ACK state from CLOSE_WAIT.

```
: goTIME_WAIT   \ *sk --
```
Move the socket into TIME_WAIT state.

```
: goCLOSING     \ *sk --
```
Move the socket into CLOSING state from FIN_WAIT1

```
: goFIN_WAIT_1  \ *sk --
```
Move socket into FIN_WAIT_1 state from SYN_RECVD or ESTABLISHED.

```
: goFIN_WAIT_2  \ *sk --
```

Move the socket into FIN_WAIT_2 state.

```
: goCLOSE_WAIT  \ *sk --
```
Move the socket into CLOSE_WAIT state after receiving a FIN in ESTABLISHED.

```
: goSYN_SENT    \ *sk --
```
Move the socket into SYN_SENT state from CLOSED.

23.10 LISTEN connection queues

From PowerNet v4.8 onwards, listening sockets save connection attempts (receipt of SYN) on a linked list of connection items. When the application, e.g. a Telnet client, wants to use a connection, it inspects the queue, and transfers the next item on the list to a new socket in SYN_RECEIVED state. The new socket waits until the socket gets to ESTABLISHED state at which point the new connection can be used to transfer data.

Compared to the previous design, listening sockets use less RAM and, because connection attempts are stored, the design is kinder to single-threaded servers.

```
struct /ConnItem        \ -- len
```
Holds unprocessed connection information after a SYN packet has been received by a listening socket. All entries are in native order.

```
: .ci           \ *ci --
```
Display the connection queue item.

```
: .cq           \ hs --
```
Display the socket's connection queue.

```
: TCPConnQ?     \ *sk -- flag
```
Return true if there is a connection on the queue.

```
: #TCPConnQ     \ *sk -- n
```
Return the number of connections on the queue.

```
: ciSame?       \ *ci1 *ci2 -- flag
```
Return true if the two connection items are for the same connection, i.e. a retry has has occurred.

```
: onConnQ?      \ *ci *sk -- flag
```
Return true if the connection is already on the socket's connection queue.

```
: >TCPConnQ     \ *tcp *sk --
```
Add an entry to a listening socket's connection queue. On failure no action is taken and we assume that the connecting socket will retry.

```
: TCPConnQ>     \ *skl *ska --
```
Take a connection from the listening socket **skl* and transfer it to the accepting socket **ska* which enters TCPS_SYN_RECEIVED state.

23.11 TCP state handlers

```
: DoTcpClosed   { *pb *sk | *ip *tcp *cb -- }
```
Handle incoming packet while in CLOSED state.

```
: DoTcpListen   { *pb *sk | *tcp *cb -- }
```
Handle incoming packet while in listening state (server).

```
: DoTcpSynReceived      { *pb *sk | *tcp *cb -- }
```

Handle incoming packet while in SYN received state (server).

`: DoTcpSynSent { *pb *sk | *tcp *cb -- }`
Handle incoming packet while in SYN sent state (client).

`: acktxbytes \ *cb nb -- nb' ; SBD011`
Removes n bytes from the transmit buffer.

`: DoTcpAcknowledge { *tcp *cb *sk | nb -- }`
Handles incoming ACK. See Wright & Stevens about ack numbers.

`: DoTcpEstablished { *pb *sk | *ip *tcp *cb tdlen -- }`
Handle incoming packet while in established state.

`: DoTcpLastAck { *pb *sk | *tcp *cb -- }`
Handle incoming packet while in last ACK state.

`: DoTcpFinWait1 { *pb *sk | *ip *tcp *cb -- }`
Handle incoming packet while in FIN_WAIT1 state.

`: DoTcpFinWait2 { *pb *sk | *ip *tcp *cb -- }`
Handle incoming packet while in FIN_WAIT2 state.

`: DoTcpClosing { *pb *sk | *tcp *cb -- }`
Handle incoming packet while in closing state.

`: DoTcpCloseWait \ *pb *sk --`
Handle incoming packet while in CLOSE_WAIT state.

`: DoTcpTimeWait { *pb *sk | *tcp *cb -- }`
Handle incoming packet while in timed wait state.

`: IncTcpPacket { *pb hs | *sk *cb -- }`
Process an incoming packet destined for hs.

`: RxTCPPacket { *pb | *ip *tcp hs *q bf -- }`
Process an incoming TCP packet.

23.12 TCP timer handling

`: DoTcpRetry? \ *cb *sk -- ; SFP021`
Send data if tx retry timer expiration.

`: DoTcpDelayedData? \ *cb *sk -- ; SFP021`
Send data if the transmit delay timer has expired.

`: DoTcpDelayedAck? \ *cb *sk -- ; SFP021`
Send ack/data if ack delay timer expiration.

`: DoTcpDisconDelay \ *cb *sk -- ; SFP021`
Handle disconnect delay timer expiration.

`: DoTcpConnectTimeout \ *cb *sk -- ; SFP021`
Handles timeout for failed connections.

`#5 tick-ms max equ TcpIdleMs \ -- ms ; SFP030`
Interval in miliseconds between TCP idle checks. Using this timer mechanism reduces the CPU load at the expense of transmission performance on local networks. Setting the value of `TcpIdleMs` to zero turns the mechanism off. The value set here in *tcp.fth* is only used if `TcpIdleMs` has not been defined in your *PNconfig.fth* file.

```
0 value TcpIdleTimer    \ -- time ; SFP030
```
Time for next TCPIDLE check.

```
: TcpIdle       { | *sk *cb -- }
```
Handles TCP idle time processing.

23.13 Primitives for the BSD layer

```
: Tcp_Open      { *sk | *cb res -- res }
```
Allows an application to open the TCP connection. N.B. This is a **blocking** function. It will **not** return until connected or timed out.

```
: Tcp_Close     { *sk | *cb res -- res }
```
Allows an application to close the TCP connection cleanly. This is the TCP primitive for the BSD `disconnect` function.

23.14 TCP initialisation

```
: TcpInit       \ --
```
Initialise the TCP layer.

23.15 Checksum test code

24 SMC LAN91C92/4/6 Ethernet Driver Code

24.1 Introduction

The file *Smc91C9x.fth* contains the hardware driver layer for the Standard MicroSystems LAN91C92/4/6 Ethernet controller chips as used on the MPE ARM Development Kit, MPE/Hiden SA1110 StrongBox and other MPE boards. The data sheets for these devices may be found at www.smsc.com and the part number for the MPE board is LAN91C96I, which can be used at 5v or 3.3v.

The controller has 64 8 bit registers organised as four banks of 16.

The code is written for portability rather than speed and uses byte accesses only. This permits the code to be used without change on big and little endian CPUs regardless of the bus interface width (8 or 16 bits).

24.2 Hardware gotchas

This code assumes a CPU with byte addressing. A cell addressed machine, e.g. most DSPs, will require considerable changes to the source code for memory buffer transfers.

When operating with fast CPUs note the following:

- The cycle time of the chip is 185ns (min. back to back access), but the read access time is only 40ns. This is detailed in the data sheet.
- At least 400ns is required between a data register access and a pointer register access. At least 400ns is required after writing the pointer register and accessing the data register. See the data sheet.
- The words `EC@` and `EC!` and the optional `EW@` and `EW!` 16 bit words by default assume that the SMC chip is memory mapped and that that the SMC memory area is contiguous. If you have a 16 or 32 bit bus and one or more low address lines are ignored you must adapt these words yourself to suit your hardware.

24.3 Configuration

The following must be defined before the file *DRIVERS\SMC91C9x.FTH* is compiled.

```
: const equ ;   \ n -- ; -- n
```

If you are using a standalone target with heads and you want interactive access to all the registers and bit masks, define `CONST` as `CONSTANT`, otherwise by default `CONST` is defined as `EQU`.

```
$50000000 const EtherBase       \ -- addr
```

Define the base address of the Ethernet controller.

```
0 equ SMC16?    \ -- flag
```

If SMC16 is non-zero, the driver will use 16 bit register accesses where possible, otherwise it will use two 8 bit accesses. SMC16 should only be set non-zero for little endian CPUs.

```
1 equ fastCPU?  \ -- n
```

Set this value false if no software intervention is required to meet the 400ns timing requirement before and after changing the pointer register.

```
0 equ smcDiags? \ -- flag
```

Set this equate true to compile diagnostic code for register dumping and so on. False by default.

```
0 equ eeprom?   \ -- flag
```

Set this equate true if the LAN91C9x has an attached EEPROM for configuration data storage. False by default.

```
0 equ GenericIP?        \ -- flag
```

Set this equate true if the Generic IP device structure defined in *ETHERCOM.FTH* is required. This is only required for systems using multiple IP devices in future releases of PowerNet.

```
1 equ sniff?    \ -- flag
```

Set this equate true to compile the packet sniffer code, which can be used to test Ethernet reception.

```
create EtherAddress     \ -- addr
```

Holds the Ethernet MAC address (six bytes). Note that you must obtain these from the IEEE (www.ieee.org) or from other sources.

```
create IpAddress        \ -- addr
```

Holds the Ethernet IP address (four bytes).

24.4 Constants

`BankSelect`
: Offset of bank select register

Bank0 Registers

`TCRL` Transmit Control Register - lo byte

`TCRH` Transmit Control Register - hi byte

`TXSTATUS` Transmit Status Register

`RCR` Receive Control Register

`MIR` Memory Information Register

`MCR` Memory Control Register

Bank1 Registers

`Configl` Configuration Register - lo byte

`Configh` Configuration Register - hi byte

`IA0` Hardware Address, Ethernet MSB

`IA1` Hardware addr

`IA2` Hardware addr

`IA3` Hardware addr

`IA4` Hardware addr

`IA5` Hardware addr

`Ctrh` Control Register - hi byte

Bank2 Registers

`MMU` Memory management unit cmd reg

`PNR` Packet number register

`ARR` Allocation result register.

`Pointerl` Memory pointer register

`Pointerh` Memory pointer register

`Datal` Reg to send packets too...

`Datah` Reg to send packets too...

`IntStatus`

`IntMask`

Misc. Constants

`RelRx` MMU Command to release memory from an rx

`RxRd` SMC Command to read received packet

`TxWr` SMC Command to write to tx area of ram

`AllocIntMask`
Memory allocated mask

`MaxMsgSize`
Max 802.3 Ether frame size in bytes

`AllocTx`

`bit0` Bitmask

`bit1` Bitmask

`bit2` Bitmask

`bit3` Bitmask

`bit4` Bitmask

`bit5` Bitmask

`bit6` Bitmask

`bit7` Bitmask

`EvenTx` Control byte for tx of even bytes

`OddTx` Control byte for tx of odd bytes

`RxTask#` Multi Task ID

24.5 Hardware Interface Layer

The default code is for a memory-mapped device at base address `EtherBase`.

```
: ec!           \ b offset --
```
Set an 8 bit register contents to *b*, in the selected bank.

```
: ec@           \ offset -- val
```
Read an 8 bit register in the selected bank.

`: ew!  \ w offset --`
For LITTLE-ENDIAN CPUs only. Set a 16 bit register contents to *w*, in the selected bank.

`: ew@  \ offset -- val`
For LITTLE-ENDIAN CPUs only. Read a 16 bit register in the selected bank.

`: ec!  \ val offset --`
VFX optimising compilers will probably produce shorter and faster code by using a compiler macro for `EC!`.

`: ec@  \ offset -- val`
VFX optimising compilers will probably produce shorter and faster code by using a compiler macro for `EC@`.

`: ew!  \ val offset --`
VFX optimising compilers will probably produce shorter and faster code by using a compiler macro for `EW!`.

`: ew@  \ offset -- val`
VFX optimising compilers will probably produce shorter and faster code by using a compiler macro for `EW@`.

`: 400ns  \ --`
For fast CPUs, use this hardware dependent word to ensure at least 400ns between back to back Ethernet chip accesses. This is required for correct operation of the pointer register. The default version here assumes 4 instructions per iteration at 5ns per instruction, and the call/return and set up overheads are ignored. See `FASTCPU?` above.

`: 400ns  \ --`
Use this version if your CPU takes at least 400 ns between back to back Ethernet chip accesses. See `FASTCPU?` above.

`: SetBank  \ bank -- ; bank = 0 to 3`
Select the active register bank.

24.6 Diagnostics

`: .bank  \ --`
Display register contents for currently selected SMC bank.

`: .reg  \ --`
Display contents of all SMC Ethernet Controller's registers

24.7 Driver Layer

`: eISR@  \ -- bmask`
Read the bank 2 interrupt status register (IST).

`: eMMU!  \ cmd --`
Send a command to the bank 2 MMU register.

`: eTCRl@  \ -- bmask`
Read the bank 0 TCRl register.

`: EnEtherTx  \ --`
Enable Ethernet transmission. Selects bank 0.

`: eSoftRst  \ --`
Initiate software reset - needed to clear Tx lock up.

`: init-SMC \ --`
Initialise SMC chip.

`: init-EtherTx \ --`
Enable Ethernet TX module.

`: init-EtherRx \ --`
Enable Ethernet RX module.

`: etheradd>CS \ addr --`
Store 6 byte Ethernet address into SMC chip from memory buffer.

`: InitEther \ --`
Perform a full initialisation of the chip, enabling Rx and Tx, and setting up the Ethernet MAC address from `ETHERADDRESS`.

`: EtherLink? \ -- flag`
Return true if the Ethernet link is established.

`: AllocTxMem \ n -- mask`
Allocate *n* bytes of memory to hold a packet for transmission. Return the contents of the ARR register, or -1 for a fatal error.

`: GetTxMem \ n -- ior`
Allocate *n* bytes of memory to hold a packet for transmission.

`: WritePacket \ addr len --`
Write an Ethernet frame to chip for transmission.

`: IsRx? \ -- t|f`
Check if incoming data is present, returning true if a packet is available for `get_ether_pkt` below.

`: DiscardRX \ --`
Throw away pending input packet due to lack of memory.

`: get_ether_pkt \ *dest maxlen -- len`
Get pending receive packet to a buffer. Note that data is valid only when `ISRX?` returns true, so that you must poll with `ISRX?` before using `GET_ETHER_PKT`.

`: IsTx? \ n -- flag`
Return true if a packet of length n can be sent.

`: send_ether_pkt \ addr len --`
Send packet from supplied buffer. The word will block until sufficient packet memory is available. Note that you should read incoming packets regularly, otherwise it is possible to get into a situation in which `SEND_ETHER_PKT` blocks for ever because there is not enough space in the device for the transmission packet. Theoretically, this should not happen because 1536 bytes are reserved for transmission, but ...

24.8 Attached EEPROM

`: ee! \ word offset --`
Write a 16 bit word into the EEPROM at EEPROM address *offset*. The EEPROM may not be present in all SMC91C9x implementations.

`: ee@ \ offset -- word`
Read a 16 bit word from the EEPROM at EEPROM address *offset*. The EEPROM may not be present in all SMC91C9x implementations.

24.9 Generic I/O for PowerNet v3 and above

The code in this section is only compiled if the equate *\fo{GenericIP?) has been defined and is non-zero.

The layout and usage of this structure is defined in the file *COMMON\ETHERCOM.FTH*.

```
create IPdevice
  ' MyInit ,       \ 0: initialisation
  ' MyTerm ,    \ 1: shutdown
  ' MyRx? ,     \ 2: receive test
  ' MyRx                \ 3: receive packet
  ' MyTx? ,     \ 4: transmit test
  ' MyTx ,              \ 5: transmit packet
  ' MyGetAddr ,    \ 6: Get device addresses
  ' MySetAddr ,    \ 7: Set device addresses
  ' MySave ,    \ 8: Save IP device state
```

`: GetAddrs      \ -- ipaddr macaddr 0`
For Generic I/O `IPDGetAddr` function

`: SetAddrs      \ ipaddr|0 macaddr|0 mode --`
Set up the Ethernet MAC and the IP addresses. At present mode is always 0, but will be used in future releases to indicate data formats. If ipaddr or macaddr are zero, the stored data will not be changed.

`create SMCvector        \ -- addr`
The device vector needed by PowerNet v3+ and *COMMON\ETHERCOM.FTH*.

`SMCvector constant IPDevice     \ -- addr`
`IPDevice` is the default device name required by *ETHERCOM.FTH*. If you have multiple ports, only one should be named `IPDevice`.

24.10 System test

This code is only compiled if the equate `SNIFF?` is non-zero.

`1536 buffer: pbuff      \ -- addr`
Buffer for SNIFF.

`: sniff         \ --`
Listens to the network and displays all the traffic that has a broadcast destination or is for this device. `SNIFF` can be used to test reception.

25 Ethernet processing task

The Ethernet ports are handled by a task which despatches PBUFs to and from other layers of the stack.

`0 equ LoseTX?  \ -- u`
When this EQU is set non-zero, every *u*th transmit packet will be discarded. This equate is used when testing the TCP retry mechanism and for simulating lossy links.

`0 equ LoseRX?  \ -- u`
When this EQU is set non-zero, every *u*th receive packet will be discarded. This equate is used when testing the TCP retry mechanism and for simulating lossy links.

25.1 Ethernet packet handlers

`: Send>Ether  \ *pb *rt -- len`
Send the data in the supplied PBuf (chain).

`params` *pb is a pbuf set to the IP layer

`returns` #bytes sent if OK else SOCKET_ERROR

`: EtherIpPacket \ *pb --`
Process a PBUF which holds a received Ethernet IP packet.

`: RxEtherPacket  \ *pbuf --`
The PBUF holds a received Ethernet packet which is processed.

`: RxEtherPkt  \ -- ; SFP001`
Get waiting packet from the Ethernet driver layer.

25.2 Link failure detection

`0 value Linked? \ --`
Returns true if PowerNet is linked to the world by an active connection.

`create LinkUpChain  \ -- addr`
Anchors the chain of words executed when the link is established. This chain must be defined at compile time.

`create LinkDnChain  \ -- addr`
Anchors the chain of words executed when the link fails. This chain must be defined at compile time.

`: AtLinkUp  \ xt --`
Add the word whose *xt* is given to link up chain. INTERPRETER word.

`: AtLinkDn  \ xt --`
Add the word whose *xt* is given to link up chain. INTERPRETER word.

`useDHCP? [if]  ' runDHCP AtLinkUp  [then]`
Start DHCP when the link comes up.

`useSNTP? [if]  ' runSNTP AtLinkUp  [then]`
Start SNTP when the link comes up.

`: checkLink  \ -- flag`
Check PowerNet's connection and set Linked?. Returns true if PowerNet is linked to the world by an active connection.

`variable CheckTime \ -- addr`
Holds the time at which the Ethernet link should next be checked.

`: nextCheck \ --`
Set the next link check time. This is normally every 200 ms.

`: CheckEther \ --`
The Ethernet link is checked every so often for an established link. If the link fails, the Ethernet task halts until the link is re-established. When it is re-established a user-extensible chain of actions takes place. This permits the system to restart actions such as DHCP and SNTP.

25.3 Ethernet task

`: DoRunEther \ --`
The task action that handles all general background Ethernet processing.

`task EtherTask \ -- addr`
The Ethernet handler task.

`: RunEtherTask \ --`
Launch the Ethernet support task.

25.4 Routing

`: (AskForEtherRoute) \ ip -- addr|0`
See if we can get a route using ARP.

`: AskForEtherRoute \ ip -- addr|0`
See if we can get a route using ARP, but retrying several times.

26 SLIP interface

The SLIP interface is handled by a task using a state machine. See SLIP\SLIPCOM.FTH.

26.1 SLIP equates

```
$C0 equ FRAME_END               \ -- char
```
Frame end character.

```
$DB equ FRAME_ESCAPE            \ -- char
```
Frame escape character.

```
$DC equ TRANS_FRAME_END         \ -- char
```
Trans Frame end character.

```
$DD equ TRANS_FRAME_ESCAPE      \ -- char
```
Trans Frame escape character.

```
#1000 equ SLIP_RX_TIMEOUT       \ -- ms
```
Receive timout in milliseconds.

26.2 Slip input functions

```
: (gotSlipChar) \ char --
```
Char is saved in the input buffer.

```
: gotTransFrameEsc      \ --
```
Received a TRANS_FRAME_ESCAPE.

```
: gotTransFrameEnd      \ --
```
Received a TRANS_FRAME_END.

```
: gotFrameEsc   \ --
```
Received a FRAME_ESCAPE.

```
: StartOfPacket \ --
```
Set up for a new packet.

```
: EndOfPacket   \ --
```
We got the end of a packet.

```
: gotFrameEnd   \ --
```
We received a FRAME_END character.

```
: GotSlipChar   \ char --
```
Handle an incoming SLIP char

```
: GetSlipChars  \ --
```
Receive characters from a SLIP port until all are in.

26.3 Slip output functions

```
: Send>Slip     { *pb *rt | ln -- int }
```
Send the data in the supplied PBuf (chain).

`*PB`	the first pbuf in the chain (only 1 allowed)
`*RT`	points to the required routing table entry
`int`	number of bytes sent if OK else SOCKET_ERROR.

`: (SendSlipChar)  \ char --`
Sends the char to the SLIP port.

`: send_frame_esc_char  \ --`
Sends the escape sequence to the SLIP port.

`: send_frame_end_char  \ --`
Sends the escape sequence to the SLIP port.

`: SendSlipChar  \ char --`
Sends the character to the SLIP port, processing escape characters.

`: SlipTx  \ *pbuf --`
Send the pbuf chain out of the slip port.

26.4 SLIP support task

`0 value SlipDevice  \ -- addr`
You MUST define a REAL serial port device for SLIP.

`: DoSlipPort  \ -- ; task action`
The action of the slip tasks.

`task SlipPortTask  \ -- addr`
Slip port task.

`: RunSlipTask  \ --`
Run the SLIP port task. Changes will be necessary to support multiple SLIP devices. The technique used by the TELNET launcher is appropriate. See SERVICES\TELNET.FTH.

27 BSD API layer

Despite its complexity, this layer is unashamedly provided to ease porting applications whose description in the TCP/IP literature is in terms of the well known BSD API. The Windows Winsock API is similar to the BSD API. Note that the PowerNet version does not provide all the facilities of the full BSD API. See *SOCKETS\BSD.FTH*.

27.1 SOCKET_ERROR returns

These words are provided to factor out error handling in the BSD layer. By default they are implemented as COMPILER macros. If implemented as discrete words, remove the comments around the "R> DROP" phrases.

`: ?se0X  \ flag -- ; flag -- SOCKET_ERROR ; exits caller`
If flag is true, returns SOCKET_ERROR and exits the CALLING WORD; otherwise does nothing. This is provided as a factor for parameter testing.

`: ?se1X  \ n flag -- n | SOCKET_ERROR ; exits caller`
If flag is true, drops 1 item, returns SOCKET_ERROR and exits the CALLING WORD; otherwise does nothing. This is provided as a factor for parameter testing.

`: ?se2X  \ n1 n2 flag -- n1 n2 | SOCKET_ERROR ; exits caller`
If flag is true, drops 2 items, returns SOCKET_ERROR and exits the CALLING WORD; otherwise does nothing. This is provided as a factor for parameter testing.

`: ?se3X  \ n1 n2 n3 flag -- n1 n2 n3 | SOCKET_ERROR ; exits caller`
If flag is true, drops 3 items, returns SOCKET_ERROR and exits the CALLING WORD; otherwise does nothing. This is provided as a factor for parameter testing.

27.2 BSD factors

These words are not part of the official BSD interface. They are either factors or useful in low-level code.

`: (pollSocket)  \ *sk -- *pb|0|SOCKET_ERROR`
Non-BSD function to poll a socket and return the next input packet buffer if available.

`: (ioctlRead)  \ *arg *sk -- 0|SOCKET_ERROR`
Return the amount of data that can be read from a socket, storing it at address **arg*.

`: (ioctlState)  \ *arg *sk -- res|SOCKET_ERROR`
Return the current socket state to the address *arg.

`: sendtoUDP  { hsocket *buffer bufflen flags | *pb res -- res }`
A factor used to send the **buffer/bufflen* block of memory by UDP.

`: SaveSktRem  \ *sk -- remip remport`
A factor to return the socket's current remote IP address and port.

`: RestoreSktRem \ remip remport *sk --`
A factor to restore the socket's current remote IP address and port.

`: recvInfo  \ *sk *name --`
A factor to extract the remote IP and port details from a received TCP or UDP packet. N.B. Replaces the now obsolete `RecvUDPinfo` and `RecvTCPinfo`. `RecvInfo` uses a socket structure as input.

```
: getSocketInfo \ hs *name *len --
```

If **len* contains at least `SOCKADDR_IN` and **name* is non-zero, **name* is the address of a `SOCKADDR_IN` structure. Copy the remote IP address and port details to `*name`.

27.3 BSD Style API

This is a loose implementation of the BSD sockets interface. It is not exactly compliant but will run most BSD style code without too many nasty surprises.

```
$04004667F constant FIONREAD            \ -- x
```

IOCTLSOCKET command code to obtain the number of bytes available.

```
$080000001 constant GET_TCPSTATE        \ -- x
```

IOCTLSOCKET command code to get the TCP state..

```
: ioctlsocket   \ hs command *arg -- res|SOCKET_ERROR
```

Perform socket operations selected by the *command* parameter. Data is returned at **arg*. The supported operations are FIONREAD and GET_TCPSTATE. FIONREAD works with both UDP and TCP sockets.

```
: pollSocket    \ hsock -- #bytes|SOCKET_ERROR
```

Non-BSD function to poll a socket and return the number of bytes available to be read. Can be used with both UDP and TCP sockets.

```
: bind          \ hs *name namelen -- res
```

Associate a socket with a family, protocol and port. The parameters **name* and *namelen* (addr/len) describe a `SOCKADDR_IN` structure. The family must be AF_INET. The port is the port number that will be listened to. The IP address is usually 0, in which case the system's IP address will be used. This allows use with systems which obtain an IP address dynamically, e.g. through DHCP.

```
: bindTo        \ hs af port ipaddr -- res
```

A non-BSD function that binds a socket to the given set of address family (af), port (port) and IP address (ipaddr). The returned result (res) is 0 for success, otherwise SOCKET_ERROR. N.B. subject to change. See `BIND`.

```
: closesocket   \ hs -- 0|SOCKET_ERROR
```

Close the socket. This is the close of last resort as it simply reclaims socket memory without performing any notification to the other end. Use `DISCONNECT` in preference.

```
: connect       \ hs *name namelen -- res
```

Open a connection to a destination defined by the SOCKADDR structure described by **name/namelen*. The adress family in the structure must be AF_INET. On success (res=0), the socket is ready to send and receive data.

```
: disconnect    \ hs -- res
```

Disconnect the socket. Returns 0 on success.

```
: DiscAllSockets        \ --
```

Disconnect **all** socket connections.

```
: socket        \ address-family socket-type proto-group -- hs|0
```

Create a new socket of the given characteristics, returning the new socket number (1..n) on success, or zero for failure. **DO NOT CHANGE** the return of `INVALID_SOCKET`=zero for failure!

```
: sendto        \ hsocket *buffer bufflen flags *name namelen -- len/err
```

A factor used to send the **buffer/bufflenlen* block of memory by TCP. If **name* is a valid SOCKADDR_IN structure then save the current remote address for the socket and override with supplied settings using CONNECT.

```
: send          \ hsocket *buffer bufflen flags -- #sent|socket_error
```

The more common form of sendto when no address override is required.

```
: recvfrom      \ hsocket *buffer bufflen flags *name *namelen -- res
```

Receive data from a socket. The parameters are as for SENDTO, **except** that **namelen* is a pointer to the length. If **name/*namelen* is a valid SOCKADDR_IN structure, on return it will contain the sender's address and port. Note that only one packet is read, and if you do not read all of it, the remaining data is **not** discarded.

```
: recv          \ hsocket *buffer bufflen flags -- len|err
```

Receive up to bufflen bytes of memory from the current packet returning the length read. Note that only one packet is read, and if you do not read all of it, the remaining data is **not** discarded.

```
: Listen        \ hs -- 0|SOCKET_ERROR
```

Starts a bound socket listening on the port specified by the previous BIND or BINDTO operation. LISTEN just changes the mode of the socket; it does not wait for anything to happen. The socket must be a TCP socket.

```
: -Listen?      \ hs -- state true | 0
```

Check the socket state, and return the state and true if it not listening. Otherwise, just return false.

```
: waitListener  \ hs -- flag
```

Wait until a connection is established or closed/closing. Return *flag=false* if established. If the socket is not established, there has been an error in the listening socket.

```
: waitConnQ     \ hs -- ior
```

Wait until the listening socket has a connection pending. Return 0 on success, or non-zero if the listening socket has failed.

```
: repSocket     \ hs1 -- hs2|0
```

Given a socket, make another of the same type using *hs1* as a template.

```
: repService    \ *sk1 *sk2 -- ior
```

create a new service data structure for *sk2* using the details in *sk1* as a template.

```
: WaitNextConn  \ hsl -- hsa 0 | reason -1
```

Wait for the next connection on listening socket *hsl*, create a socket and service for the connection, and return the new socket *hsa* and 0. If a failure occurs, return *reason* and -1. If the listening socket failed *reason* is non-zero.

```
: Saccept       \ hs *skaddr *sklen -- hsnew|SOCKET_ERROR
```

The PowerNet version of the BSD **accept()** call is named Saccept to prevent a name conflict with the input routine accept. Wait until a connection has been made to the listening TCP socket *hs* or there is an error in the socket. If a connection has been established, transfer it to a new socket *hsnew*. If **skaddr* is non-zero the contents of **sklen* are at least SOCKADDR_IN, **skaddr* is filled in with the IP address and port of the remote end.
After a successful operation, the new socket is used for the connection just made, and the old socket remains listening.

If SOCKET_ERROR is returned, no new socket is available and previous socket *hs* should be checked to see if it should be closed and remade.

27.4 Extensions

These extensions make life a bit simpler when connecting to servers.

`: TCPsocket \ -- hs|0`
Create a TCP socket. Return the socket handle on success or `INVALID_SOCKET` (0) on error.

`: connectTo \ caddr u port# socket -- socket ior`
Attempt to connect to a server. The socket has already been created in the appropriate mode. The value of *socket* must be a positive non-zero number. *Caddr/u* describes the server address either as a name or an IPaddress string and *port#* is the requested port. If *u* is zero, *caddr* is treated as a 32 bit number representing an IPv4 address. In this implementation, *u* must be zero. On success, the socket and zero are returned, otherwise SOCKET_ERROR and an error code are returned.

`: TCPConnect \ c-addr u port# -- socket ior`
Attempt to create a TCP socket and connect to a server. *Caddr/u* describes the server address either as a name or an IPaddress string and *port#* is the requested port. If *u* is zero, *caddr* is treated as a 32 bit number representing an IPv4 address. In this implementation, *u* must be zero. On success, the socket and zero are returned, otherwise SOCKET_ERROR and an error code are returned.

`: UDPConnect \ c-addr u port# -- socket ior`
Attempt to create a UDP socket and connect to a server. *Caddr/u* describes the server address either as a name or an IPaddress string and *port#* is the requested port. If *u* is zero, *caddr* is treated as a 32 bit number representing an IPv4 address. In this implementation, *u* must be zero. On success, the socket and zero are returned, otherwise SOCKET_ERROR and an error code are returned.

28 PowerNet diagnostic tools

```
: check_tcp_cksum       { *ip | *tcp buff[ #12 ] -- flag }
```
Check incoming TCP cksum, returning true for a good checkum.

```
: dump_line     \ address bytes --
```
Display a small memory area as a single line of bytes followed by the ASCII equivalent.

```
: .protocol     \ protocol --
```
Display the protocol type.

```
: .socket_type  \ type --
```
Display the socket type.

```
: .iphdr        \ addr --
```
Display an ip header.

```
: .udp          \ *ip --
```
Prints UDP packet information.

```
: .tcpFlags     { fl -- }
```
Display flag word contents.

```
: .tcp          { *ip | *tcp -- }
```
Prints TCP packet information.

```
: .tcpcb        { *cb -- }
```
Prints TCP packet information.

```
: .sockaddr_in  \ ^sockaddr_in --
```
Display the contents of a SOCKADDR_IN or SOCKADDR structure.

```
: show-socket   \ hs --
```
Short form of display of socket data.

```
: netstat       \ --
```
Display short-form data about all non-closed sockets.

```
: .ippkt        { *ip -- }
```
Prints IP packet information.

```
: .etheradd     \ addr --
```
Display an Ethernet MAC address at addr.

```
: .route        \ *rt --
```
Show the routing table entry contents.

```
: .routes       \ --
```
Show all routing table entries.

```
: .qd           \ *q --
```
List the queue data.

```
: (.socket)     \ *sk --
```
Display the supplied socket structure.

```
: .socket       \ HSOCKET --
```
Long form display from the supplied socket handle.

```
: .k    { | *sk -- }
```

Show socket states in short form.

```
: .lk           \ --
```
Show all active socket states in long form.

```
: .err          \ --
```
Display current socket error.

```
: z             \ --
```
Dump 64kb of memory at 0000:0000 and time the result. Mostly used for Telnet checking and performance testing.

```
: zz            \ --
```
Perform Z above until a key is pressed.

29 TFTP receiver

The TFTP server receives files from a remote client. Only one connection at a time is supported. See SERVICES\TFTP.FTH.

In order to use the TFTP server, you must define the actions required of the TFTP events.

N.B This is alpha test code.

29.1 Ident Block

```
create $MODULE  \ -- addr
```
Module name string.

```
create $VERSION \ -- addr
```
Version string.

```
create $COPYRIGHT       \ -- addr
```
Copyright string

29.2 Global data

```
0 VALUE TFTP_Socket     \ -- n
```
Socket used by TFTP server.

```
#512 VALUE TFTP_BLOCKSIZE       \ -- n
```
TFTP data block size.

```
TFTP_BLOCKSIZE 4 + VALUE TFTP_MAXPACKETSIZE     \ -- n
```
TFTP data packet size.

```
0 VALUE *TFTP_PACKET    \ -- addr
```
Current TFTP packet address.

```
0 VALUE TFTP_PACKETSIZE \ -- n
```
Current TFTP packet data block size.

```
0 VALUE blockid \ -- n
```
If blockid is non-zero then we are connected.

```
STATE_IDLE VALUE TFTPState      \ -- n
```
Current State Machine state ID.

29.3 TFTP State Machine equates

29.4 Event action place-holders and defaults

The reciver uses these three DEFERred words to handle incoming TFTP events.

```
defer TEVENT::BeginReceive      \ c-addr u -- okay?
```
Process a STARTRX (upload to server) request.

```
defer TEVENT::RxData            \ addr len --
```
Process a receive data request. The parameters describe the TFTP data block.

```
:NONAME         \ c-addr u -- okay?
```

Default to handle the string that comes with the STARTRX (upload to server) request. Just display the string and return true to carry on.

```
:NONAME         \ addr len --
```
Default to process a received data block. Just show the address and length.

29.5 Utility Words

```
: DiscardTFTP   \ --
```
Discard the current TFTP packet.

```
: AbortTFTP     \ c-addr u --
```
Display the TFTP error message and discard the current packet.

```
: zcount        \ zaddr -- zaddr len
```
A version of COUNT for zero terminated strings, returning the address of the first character and the length.

29.6 TFTP State Handlers

```
: smIDLE        { | namelen -- }
```
The TFTP system is idle. Perform a lot of error checks and report them if the equate DIAGS? is true.

```
: smSTARTRX     \ --
```
Handles start of receiving a file.

```
: smSTARTTX     \ --
```
Handles start of a (refused) transmission request.

```
: smFINISHEDRX  \ --
```
Handles the end of file reception.

```
: smFINISHEDTX  \ --
```
Handles the end of file transmission.

```
: smRXDATA      \ --
```
Handles a received data packet.

```
: smTXDATA      \ --
```
Handles transmission of a data packet.

```
: smACKDATA     \ --
```
Handles a data acknowledge.

```
: smWAITFORACK  \ --
```
Handles waiting for a data acknowledge.

TFTP Server Task

```
task TFTPTask   \ -- addr
```
The TFTP task data structure.

```
: TFTP_Install  \ -- flag
```
Starts the TFTP server, returning true for success or false for failure. SUBJECT TO CHANGE.

29.7 Event Action Handlers

This code is currently commented out. Read it as an application example.

30 Support for TCP services

PowerNet supports multithreaded TCP servers that can accept multiple connections. The model used is that one task listens on a port, establishes a connection, passes the socket to a service task to run the service, and then creates a new listening socket. When a connection to a service task is no longer established, a support task cleans up the service specific data, closes the service socket, and terminates the service task.

The focus of the tools is for servers using text, e.g. HTTP and Telnet. To enable use of standard Forth words and parsing tools, a Generic I/O device and a `TIB` are established for each connection .

From PowerNet v4.6 onwards and then again in v4.8, much attention to reducing RAM usage has taken place. This comes at the expense of performance. Use of the the low RAM configuration may be necessary for single-chip applications. From v4.8 onwards, you can use the equates `SVlowRAM?` and`SVsingle?` in the PowerNet configuration file to reduce RAM consumption in listen state by up to 2kb per listening socket above the reductions already achieved in the TCP layer.

30.1 Service numbers

These are MPE defined for use in the socket structure.

```
0 equu service_none      \ -- n
```
Defines a null service.

```
1 equ service_Telnet     \ -- n
```
Defines a Telnet service.

```
2 equ service_HTTP       \ -- n
```
Defines an HTTP service.

```
3 equ service_FTP        \ -- n
```
Defines an FTP service.

```
8 equ service_Echo        \ -- n
```
Defines an Echo service.

```
9 equ service_MultiChat \ -- n
```
Defines a MultiChat service. Windows version only.

```
10 equ service_ModBus
```
Defines a ModBus TCP service.

```
20 equ service_App
```
Service number for an application-specific service.

```
: .service      \ n --
```
Display the service type corresponding to n.

30.2 Service specific data

These data definitions are required by each server task. The data is allocated at the start of the task and released when the task is `TERMINATE`d. The chain `SVchain` links all the service tasks.

Each task has `USER` variables `MY_SOCKET` and `MY_HSOCKET` which hold the socket address and socket number. From these, the service specific data can be found.

The first part of a service data area is common to all services, and additional fields can be added as required. See *TELNET.FTH* and *HTTP.FTH* for examples. The common data area includes service managment data and facilities for routing input and output through the normal Forth `KEY` and `EMIT` wordset.

`variable SVchain \ -- addr`
Holds the tail of the service chain.

`#64 equ /SVOB \ -- n`
Size of a service output buffer. A value of 64 or 128 bytes is sufficient to provide satisfactory output for Telnet.

`#64 equ /SVIB \ -- n`
Size of a service input buffer.

`#256 equ /SVtib \ -- n`
Size of the service's TIB area.

`struct /SVdata \ -- len`
Defines the common data in a service specific data area.

```
  int SVlink                    \ link to previous service
                                \ N.B. MUST BE FIRST
  int SVtask                    \ task that runs this service
  int SVsk#                     \ socket# used by this service
  int SVsendflags               \ TCP override flags for lower layers
  int SVdone                    \ set nz to close service
  int SVappClean                \ *sv -- *sv ; application specific clean up xt
dup equ /CSVdata                \ size of core service data
\ end of core structure
  /SVtib field SVtibBuff        \ service TIB, e.g. for Telnet
dup equ /MSVdata                \ size of core+TIB service data
\ end of core plus input buffer
SVlowRAM? 0= [if]
  int #SVOB                     \ number of characters in SVOB ; SFP002
  /SVOB field SVopBuff          \ service output buffer ; SFP002
  int #SVIB                     \ number of characters in SVIB
  int ^SVIB                     \ offset of next character in SVIB
  /SVIB field SVipBuff          \ service raw input buffer
[then]
end-struct
```

The field `SVappClean` contains the xt of a word

```
  cleanFTP ( *sv -- *sv )
```

that frees any aqdditional resources that are not released by free the service data area. For an example, see `cleanFTP` in SERVICES/FTP.fth.

`: MySVD \ -- addr`
Returns the address of the task's service data.

```
: SVTib          \ -- tib
```
Returns the address of the TIB buffer for this task.

```
: SVdone?        \ -- flag
```
Return true if the socket can be closed.

```
: SVbye          \ --
```
Set the `SVdone` exit flag.

30.3 Server assistance

```
: CheckSV        \ -- flag
```
Run by the service tasks to check whether the service task should be closed. Flag is returned true if the service task should be closed.

30.4 Service KEY, EMIT and friends

```
$0A equ AcceptChar       \ -- char
```
The character returned by`SVkey` below when an error has occurred. This should be the character that `SVaccept` uses as a line terminator, usually LF.

30.4.1 Low RAM version

In this version all buffering is performed in the socket layer. This saves RAM at the cost of performance. On a local area network the penalty may be significant - it certainly is for Telnet.

```
: SVkey?         \ -- flag
```
Return true if a character is avilable from the service's client or an error has occurred.

```
: SVkey          \ -- char ; receive char
```
Return a character from the service's client. If the service's `SVdone` flag is set, an LF is returned.

```
: SVtype         \ caddr len --
```
Send a string/buffer to the service's client.

```
: SVemit         \ char --
```
Send a character to the service's client.

```
: SVcr           \ --
```
Send a CR/LF pair to the service's client.

```
: SVflushOP      \ --
```
A compatibility `NOOP` in the low RAM version.

30.4.2 High performance version

```
: SVIBuffer      \ -- addr
```
Returns the address of the input buffer for this task.

```
: SVOBuffer      \ -- addr
```
Returns the address of the output buffer for this task.

```
: isSVInput      \ --
```
Read any available characters from the incoming TCP stream into the service input buffer.

```
: (SVkey?)       \ -- flag ; check receive char
```
Return true if a character has been received by the server or the service must be closed.

```
: SVflushOP      \ -- ; SFP002
```

Send current buffer if not empty by passing it to the socket. `SVflushOP` is used by `SVkey?` and `SVkey` so that pending output is transmitted. If your code does not call either of these, e.g. through the Generic I/O, you should add `SVflushOP` to your code where appropriate.

```
: (SVemit)      \ char -- ; SFP002
```

Send a character to a client of this service.

Service vectored I/O

```
: SVkey?        \ -- flag ; check receive char
```

Return true if a character has been received by the server. Any pending output characters are sent.

```
: SVkey         \ -- char ; receive char
```

Return a character from the service's client. Any pending output characters are sent first. If the service's `SVdone` flag is set or a socket error has occurred, an LF character is returned.

```
: SVemit        \ char -- ; emit char
```

Send a character to a service's client.

```
: SVtype        \ caddr len -- ; display string
```

Send a string to a service's client

```
: SVcr          \ -- ; display new line
```

Send a new line sequence to a service's client

30.4.3 Generic I/O device

```
create ConsoleSV        \ -- addr ; OUT managed by upper driver
```

Function despatch table for service I/O. `OUT` is managed by the upper level driver.

```
: Console=SV    \ --
```

Select the service I/O as the console.

```
: Init-ConsoleSV        \ --
```

Initialise for console I/O by the service. Note that the service's socket must have been set up and and the private service area initialised.

30.4.4 Service console support

```
: +SV_responsive        \ --
```

Set the socket to use the TCP_PSH flag when sending to improve interactivity.

```
: -SV_responsive        \ --
```

Set the socket not to use the TCP_PSH flag when sending.

```
: SV_responsive?        \ -- res
```

Return non-zero if the socket's TCP_PSH flag override is set.

```
: SVaccept      \ c-addr +n1 -- +n2 ; read up to LEN chars into ADDR
```

Read a string of maximum size n1 characters to the buffer at c-addr, returning n2 the number of characters actually read. Input is terminated by LF, and CR is ignored. This satisfies the requiresments of DOS, Windows, Unices and the TCP/IP NVT (Network Virtual Terminal). If `ECHOING` is non-zero, characters are echoed. If `XON/XOFF` is non-zero, an XON character is sent at the start and an XOFF character is sent at the the end.

```
: SVquery       \ -- ; fetch line into TIB
```

Reset the input source specification to the console and accept a line of text into the input buffer.

30.5 Service creation and deletion

`: isMySocket \ hs --`
Set up `USER` variables for this socket.

`: NoSocket \ --`
Zero the `USER` variables `MY_HSOCKET` and `MY_SOCKET`.

`: ShutErrSocket \ -- socket_error`
Close the current server socket and return `SOCKET_ERROR`. All allocated socket and service memory is released by the close.

`: InitServerSocket { #service /data #port -- res }`
Create a listening socket and the private data for a TCP service conversation. On success, the `USER` variables `MY_HSOCKET` and `MY_SOCKET` contain the socket number and socket address. On failure, these variables contain zero.

`#service` service type number.

`/data` size of the service private data area, at least /SVDATA.

`#port` port number to listen on.

`res` socket number (1..n) or SOCKET_ERROR.

`: SVinitiate \ xt -- task|0`
Allocates memory for a server task and `INITIATE`s it with the given xt, returning the task's TCB address. If memory cannot be allocated, zero is returned.

`: (SVterminate) \ task --`
Terminates a server or service task and frees off the task memory. Does not `PAUSE`

`: SVterminate \ task --`
Terminates a server or service task and frees off the task memory.

30.6 Service listening task

`: .SVmessage \ caddr len --`
Display a message to the current output (usually the console) in the form:

```
<service> <given text> on socket <n>
```

The `USER` variables `MY_SOCKET` and `MY_HSOCKET` must contain valid data.

`: ServiceCreate \ #service /data port# --`
Create a new service listening socket.

`: StartService \ hs xt --`
Initialise and launch a new service task on socket *hs*. The action of the new task is given by *xt* given by xt.

30.7 Service support tools

`: waitSocketSent \ hs --`
Wait until all transmit data for a socket has been sent and acked.

`: wait-socket-empty \ --`
Wait until all output has been sent from the socket.

`: SVdisconnect \ --`
`Disconnect` the current service socket and run `SVbye`.

`: SVstartup \ --`
Performs the default actions when a service starts. I/O is set to the console, `BASE` to decimal, and a console message is issued.

`: SVshutdown \ --`
Performs actions required when a service finishes.

`: SRVRstartup \ caddr len --`
The first action a server task should perform. The string is displayed on the console with a startup message.

30.8 Service output

In order to avoid race conditions, a separate task handles testing whether a service should be closed.

`variable SVkillChain \ -- addr`
Holds the tail of service tasks to be destroyed.

`: ?SVkill \ *SVdata --`
If the service has a task, i.e. is a service rather than a server, move it to the kill chain, otherwise just free the memory.

`: ?ServiceClose \ --`
Close any service tasks with a non-zero `SVDONE` field in the private service area.

`: ?ServiceKill \ --`
Kill any service tasks that are on the kill chain and free the task and service memory. The kill chain is extended by `CloseSocket`. The service task clean up action (the xt in the `SVappClean` field) is run before memory is freed.
The field `SVappClean` contains the xt of a word

```
cleanFTP ( *sv -- *sv )
```

that frees any aqdditional resources that are not released by free the service data area. For an example, see `cleanFTP` in SERVICES/FTP.fth.

`: ServiceIO`
Handle any queued output - called in the service task.

30.9 Diagnostics

Diagnostic code is only compiled if the `EQU`ate `DIAGS?` is set non-zero.

31 TCP Echo socket

The Echo server just returns anything it receives to the sender. It is a simple test of TCP transmission and reception. Only one connection at a time is supported.

To save code space, the Echo server uses the server system in *SERVICES\Servers.fth* and `Saccept` from the BSD layer.

As discussed in the Telnet chapter, HyperTerminal PE is a reasonable test client.

`TCP_PSH value EchoFlags \ -- n`
For fast response for use with an interactive terminal set this to TCP_PUSH. For fast bulk response, set this to zero.

`7 equ EchoPort# \ -- n`
Port on which Echo server listens. Moved to *PNconfig.fth*.

`$100 equ EBLen  \ -- n`
Size of the Echo receive buffer.

`EBLen buffer: EchoBuffer        \ -- addr`
The Echo receive buffer.

`#60000 equ /EchoMs      \ -- ms`
Max delay after a connection before the socket is closed if there is no transmission.

`: echoEst?      \ hs -- flag`
Return true if the socket is in ESTABLISHED state.

`: echoWait      \ hs -- flag`
Wait for something to be received up to the timeout period. *Flag* is returned true if the socket is still ESTABLISHED and there has been no timeout.

`: echoResponse  \ hs --`
Read the input and respond.

`: echoClose     \ hs --`
Close the echo connection.

`: EchoService   \ hs --`
Service an Echo connection.

`: DoRunEchoSocket       \ --`
The Echo task.

`task TcpEchoTask        \ -- addr`
The task running the Echo socket.

`: RunEchoSockTask       \ --`
Start the Echo task.

32 Telnet Server

The Powernet Telnet server is a multithreaded server that can accept multiple Telnet connections. Local echo is not required on the client. The model used is described in *SERVERS.FTH*.

For testing, please be aware that the standard Windows Telnet client is very slow. A nuch faster alternative is HyperTerminal Personal Edition from:

```
http://www.hilgraeve.com
```

32.1 Telnet specific data

These data definitions are required by each Telnet server task. The data is allocated at the start of the task and released when the task is `TERMINATE`d. The chain `SVCHAIN` links all the service tasks.

```
#5023 equ TelnetPort#   \ -- n ; standard is 23
```
Define the port used for the Telnet server. The standard port is 23, but 5023 is the default set for PowerNet as most application Telnet servers are private. Moved to *PNconfig.fth*.

32.2 IAC handling

Not yet implemented.

```
: DoEscapeSequence      \ addr -- n
```
Process a character sequence with the TNET_IAC escape character at addr, returning n = decoded length. At present these sequences are ignored.

```
: ParseTelnetBuffer     \ numchars --
```
Process incoming Telnet data.

32.3 Telnet vectored I/O

The Telnet server establishes its own generic I/O based on that in *SERVERS.FTH* in order to handle IAC processing in the future.

```
create ConsoleTN        \ -- addr ; OUT managed by upper driver
```
Function despatch table for Telnet I/O. `OUT` is managed by the upper level driver.

```
: Init-ConsoleTN        \ --
```
Initialise for console I/O by Telnet. Note that the Telnet socket must have been set up and and the private service area initialised.

```
/SVdata equ /TNdata     \ -- len
```
Required size of console service area.

```
: Console=Telnet        \ --
```
Select Telnet as the console.

32.4 Telnet service tasks

```
: tn_announce   \ --
```
Issues Telnet signon message.

```
: (tn_login)    \ -- ior
```

Perform the default login procedure and return non-zero if successful so that the return code can be used as a user identifier by higher level code if required

`defer tn_login  \ -- ior`
Perform the assigned login procedure and return non-zero if successful so that the return code can be used as a user identifier by higher level code if required. When zero is returned for unsuccessful login, the Telnet session is terminated. See `TelnetService`.

`: tn_quit       \ --`
Empty the return stack, store 0 in `SOURCE-ID`, and enter interpretation state. `TN_QUIT` repeatedly `SVaccept`s a line of input and `INTERPRET`s it, with a prompt if interpreting and `ECHOING` is on. Note that any task that uses `TN_QUIT` must initialise `'TIB`, `BASE`, `IPVEC`, and `OPVEC`. Note that `TN_QUIT` clears the stack on exit.

`: TNbye         \ --`
Use this to close the connection in Telnet.

`: TelnetService \ --`
The Telnet service task launched for each established Telnet connection.

32.5 Telnet listening task

`: TNserverPass  \ --`
One iteration through the Telnet server.

`: TelnetServer  \ -- ; stay here forever`
The Telnet listening task.

`: TelnetServer  \ -- ; stay here forever`
The Telnet listening task.

`0 value TelnetTask      \ -- 0|task`
Returns 0 or the Telnet server task if running.

`: RunTelnetSockTask     \ --`
Start the Telnet server task.

`: StopTelnet    \ --`
Stop the Telnet server.

32.6 Diagnostics

This code is only compiled if the `EQU`ate `DIAGS?` is set non-zero.

`: .TelnetChain  \ --`
Display data about the Telnet sockets.

`: .tn           \ -- ; synonym`
Display data about the Telnet sockets.

33 FTP Server

The Powernet FTP server is a multi-threaded server that can accept multiple FTP connections. The model used is described in *SERVERS.FTH*. The code for the command channel is based on the Telnet code.

The objectives of the design are to use "not a lot" of RAM, and to keep the code size down by not providing many bells and whistles. A side effect of the low RAM usage is that directory listings are rather slow. Initially, the intention was only to support the requirements of RFC959, the primary FTP specification. The only client that actually works fully in this mode is WS_FTP. In order to support clients that are not so forgiving, a few extra commands are supported. The FTP server has been tested with:

- Ipswitch WS_FTP Pro (2007). This has been our Windows FTP client of choice for many years. Windows only.
- WinSCP. A functional free tool. Windows only.
- FileZilla. A functional free tool. Windows, Linux and Mac OS X.

The use of the *FATfiler* file system code is assumed. Because this is not a fully thread-safe file system and only has a single "working directory" for all threads, this FTP server is not suitable for use with FTP clients that assume a Unix-style operating system. Such clients include the FTP client built into Finder on Mac OS X. FileZilla works well on the Mac, as well as on Linux and Windows.

33.1 FTP data

`struct /FTPdata \ -- n`
The standard service data structure is extended for FTP.

`: cleanFTP \ *sv -- *sv`
Clean up an FTP service block before it is released.

`: my_ftpState \ -- addr`
Holds the state of the FTP system.

`: my_ftpPassive? \ -- addr`
Holds the passive flag.

`: my_ftpBinary? \ -- addr`
Holds the binary flag.

`: my_ftpQuit? \ -- addr`
Holds the QUIT flag.

`: my_ftpDataIP \ -- addr`
Holds the IP adddress of the data channel.

`: my_ftpDataPort \ -- addr`
Holds the port number of the data channel.

`: my_ftpDataSock \ -- addr`
Holds the socket handle of the data channel.

`: my_ftpDataState \ -- addr`
Holds the state number of the data channel.

```
: my_ftpDataFile        \ -- addr
```
Holds the handle of the file being read/written.

```
: my_ftpLine?           \ -- addr
```
Holds true if a complete line can be processed.

```
: my_ftpLineLen         \ -- addr
```
Holds the current length of the line excluding terminators.

```
: my_ftpSrc$            \ -- addr
```
Buffer that holds a source file path as a counted string.

```
: my_ftpDest$           \ -- addr
```
Buffer that holds a destination file path as a counted string.

```
: my_ftpBuff            \ -- addr
```
Buffer that holds FTP data for transfer.

33.2 FTP vectored I/O

The FTP server establishes its own generic I/O for the data and command channels. The data channel operations are written for minimum RAM usage and use the sockets for buffering.

33.2.1 Data socket

```
: checkSocket   \ hs -- ior
```
The *ior* is returned non-zero if the socket is invalid or not in TCPS_ESTABLISHED state.

```
: sockKey?      \ hs -- #chars
```
Return the number of available characters from the socket. If an error has occurred, 1 is returned.

```
: sockKey       \ hs -- char
```
Return a character from a socket. On a socket error, an LF is returned.

```
: sockType      \ caddr len hs --
```
Send a string/buffer to the socket.

```
: sockEmit      \ char hs --
```
Send a character to the socket.

```
: sockCr        \ hs --
```
Send a CR/LF pair to the socket.
The following five words are used to provide generic I/O on the FTP data channel.

```
: FTPdataKey    ( -- char )     my_ftpDataSock @ sockKey  ;
: FTPdataKey?   ( -- flag )     my_ftpDataSock @ sockKey?  ;
: FTPdataEmit   ( char -- )     my_ftpDataSock @ sockEmit  ;
: FTPdataType   ( caddr len -- ) my_ftpDataSock @ sockType  ;
: FTPdataCr     ( -- )          my_ftpDataSock @ sockCr  ;

create ConFTPdata       \ -- addr ; OUT managed by upper driver
```
Function despatch table for FTP data channel I/O. `OUT` is managed by the upper level driver.

```
: [FTPdataIo    \ -- ; R: --  ipvec opvec
```
Redirects console I/O to the FTP data channel. Use in the form:

```
  [FTPdataIo ... io]
```

33.2.2 Command socket

```
create ConsoleFTP       \ -- addr ; OUT managed by upper driver
```

Function despatch table for FTP command channel I/O. `OUT` is managed by the upper level driver.

```
: Init-ConsoleFTP       \ --
```

Initialise the command channel. Note that the FTP socket must have been set up and and the private service area initialised.

```
: FTPio         \ --
```

Select the FTP command channel as the console.

```
: checkFTP      \ -- ior
```

Return non-zero if there is an error in the FTP command channel.

33.3 Sampling the command channel input

```
: -ftpLine      \ --
```

Reset the FTP line input.

```
: +ftpLine      \ --
```

Mark that a complete line is available.

```
: ftpBS         \ --
```

The backspace operation for input.

```
: +ftpCmdChar   \ char --
```

Add the character to the command line being assembled.

```
: ?FTPacceptable        \ --
```

Process the next service input character on the command channel. Input is terminated by LF, and CR is ignored. This satisfies the requirements of DOS, Windows, Unices and the TCP/IP NVT (Network Virtual Terminal).

33.4 Diagnostic control

```
1 value FTPdiags?       \ -- n
```

Set this non-zero to get diagnostic information.

```
: [ftp  ( -- )  FTPdiags? if  [io consoleio decimal  ;
```

A `COMPILER` macro used to surround debug code, and terminated by `FTP]`.

```
[FTP  ." debug message"  FTP]
```

```
: ftp]  ( -- )  io]  endif  ;
```

A `COMPILER` macro that terminates an `[FTP ... FTP]` structure.

```
: .fdLine       \ caddr len --
```

Display FTP text with leading CR on Forth console. If `FTPdiags?` is set to zero, no action is taken.

33.5 Directory listing for FTP

Each line of the display is "sort of" in Unix ls format.

```
----------   1 owner    group         1803128 Jul 10 10:18 ls-lR.Z
d---------   1 owner    group               0 May  9 19:45 Softlib
```

`create months   \ -- addr`
String containing 3 character text for the months.

`: .ftpDate      \ --`
Display the current directory entry's date in the form:

```
  Mmm dd  yyyy
```

e.g.

```
  Apr 30  2012
```

`: .ftpDirLine   \ --`
Display a directory entry in FTP format. FTP clients get the size information from this format.

`: .ftpFile      \ --`
List the file data for the last file found.

`: .ftpdir       \ --`
Display a list of files in FTP format.

`: .ftpDirNlst   \ --`
Display a list of file names in FTP NLST format.

33.6 Status returns

`: .ftpResp      \ caddr len --`
Send the string plus a CR/LF pair to the command socket and optionally to the console.

`: .ftp150       \ --`
Return status 150 - about to transfer data

`: .ftp200       \ --`
Return status 200 - good command.

`: .ftp202       \ --`
Return status 202 - not needed

`: .ftp211       \ --`
Return status 211 - not available

`: .ftp230       \ --`
Return status 230 - user logged in.

`: .ftp250       \ --`
Return status 250 - good file command.

`: .ftp226       \ --`
Return status 226 - transfer successful.

`: .ftp331       \ --`
Return status 331 - password needed

`: .ftp350       \ --`
Return status 350 - need more info.

`: .ftp425       \ --`
Return status 425 - can't make data connection.

```
: .ftp426       \ --
```
Return status 426 - aborted.

```
: .ftp450       \ --
```
Return FTP error code 450 - action not taken.

```
: .ftp451       \ --
```
Return FTP error code 451.

```
: .ftp502       \ --
```
Return FTP error code 502 command not implemented.

```
: .ftp504       \ --
```
Return FTP error code 504.

```
: .ftp550       \ --
```
Return FTP error code 550.

33.7 Data socket operations

```
: [sm           \ -- 0
```
Starts the definition of a state machine's states.

```
: smState       \ n -- n+1
```
Defines the next state as an `EQU` and increments the state number.

```
: sm]           \ n --
```
Finishes the state machine and defines an equate of the number of states.
The FTP data transfer state machine.

```
[sm
  smState ftpDtIdle             \ No data transfer in progress
  smState ftpListening          \ PASV mode, wait connection
  smState ftpConnected          \ wait transfer command
  smState ftpReadSendData       \ RETRieve
  smState ftpRecvWriteData      \ STORe
  smState ftpErrorState         \ it's bad if we get here!
sm] #FTPsm      \ -- n
```

Select the the FTP data socket states.

```
: goFTPidle     \ --
  ftpDtIdle my_ftpDataState !  ;
: goFTPlistening        \ --
  ftpListening my_ftpDataState !  ;
: goFTPconnected        \ --
  ftpConnected my_ftpDataState !  ;
: goFTPReadSend \ --
  ftpReadSendData my_ftpDataState !  ;
: goFTPRecvWrite        \ --
  ftpRecvWriteData my_ftpDataState !  ;


: umin          \ u1 u2 -- u1|u2
```
Minimum of two unsigned values.

```
: closeDataFile \ --
```

Perform an emergency close of the data file if it is open.

`: hasConn?  \ hs -- ior true | 0`
Return true and an ior if the socket has an error or a completed connection. For a good connection, *ior* is non-zero.

`: newDataSocket \ -- hs|0`
Create a new FTP data socket and set it to listen. The socket handle is also stored in the FTP service structure.

`: discDataSocket  \ --`
If open, flush the socket transmit data and disconnect the FTP data socket. This is a graceful close unless the socket has failed.

`: closeDataSocket  \ --`
If open, close the FTP data socket. This is not a graceful close.

`: termDataStream  \ --`
Gracefully shut down a STREAM mode data transfer and return to FTPDTIDLE state.

`: closeDataStream  \ --`
Hurriedly shut down a STREAM mode data transfer and return to FTPDTIDLE state.

`: CheckFTPdata  \ -- ior`
Return non-zero if the FTP data socket is in error.

`: FTPdataFailed?  \ -- ior`
Return non-zero if the FTP data socket is not in established state. If not established, the data socket is closed and we return to FTPDTIDLE state.

`: doDTidle  \ --`
The action in FTPDTIDLE state. Check the data socket. If it fails, close it.

`: doDTlistening \ --`
The action in FTPLISTENING state.

`: doDTconnected \ --`
The action in FTPCONNECTED state.

`: doDTReadSend  \ --`
The action in FTPREADSENDDATA state.

`: doDTRecvWrite \ --`
The action in FTPRECVWRITEDATA state.

`: doDTerror  \ --`
The action in FTPERRORSTATE state.

`create FTPdataActions  \ -- addr`
A table of xts corresponding to the FTP data channel state.

`: doFTPdata  \ --`
We have a data socket. Execute the action for the state.

`: waitDataConnected  \ -- ior`
Wait for a passive mode data connection to be made. Return non-zero on failure.

`: waitDataEstablished  \ -- ior`
Wait for a passive mode data connection to get to TCP state `TCPS_ESTABLISHED`. Return non-zero on failure.

33.8 Command processing

An FTP command exists on a single line. The first token identifies the command. Each command identifier is a Forth word which parses any more data needed by the command.

`: parse-name \ "text" -- caddr len`
Return the next space-delimited string from the input stream.

`: parse-path \ "<pathname>" -- caddr len`
Return the next space-delimited path name from the input stream. Clip it to `MAX_PATH 1-` bytes.

`: getSrcParam \ --`
Read the parameter and store it as the source name.

`: getDestParam \ --`
Read the parameter and store it as the destination name.

`: getSrcDir \ --`
Read the parameter and store it as the source name. If the name ends in a '/' character, remove it. Some FTP clients terminate directory names with a '/', which can confuse the FAT file system.

`: SrcParam \ -- addr len`
Return the string saved as the source parameter.

`: FTPannounce \ --`
Issues FTP signon message.

`: +decByte \ u caddr len -- u' caddr' len'`
Accumulate the next byte of a comma separated set of numbers, e.g. 1,25,33,4. The accumulator u is shifted left by 8 bits and the next number in the text is added. The updated accumulator and remaining text are returned.

`: .decByte \ u -- u'`
Display the top byte of u as a decimal number and shift it left by 8 bits.

`: .comma \ --`
Display a comma.

`: FTPabort426 \ -- 426`
Close the data socket and data file and command with a 426 response.

`: setDataSock \ -- ior`
Depending on the mode, check or establish a data connection. Return non-zero on error.

`: .quo ( -- ) [char] " emit ;`
Display a double quotes.

`: .pwdResp \ --`
Display the PWD response with no CR/LF.

`: .257SrcResp \ --`
A 257 response with the source buffer

33.9 Login and security

Access to FTP is provided through the FTP verbs (words) `USER` `PASS` and optionally `ACCT`.

`defer doFTPuser \ --`
Process the FTP USER command. The default accepts any user. Once the user and password have been confirmed, the `ftpState` variable can be set to 1.

`defer doFTPpass \ --`
Process the FTP PASS command. The default accepts any password. Once the user and password have been confirmed, the `ftpState` variable can be set to 1.

`defer doFTPacct \ --`
Process the FTP ACCT command. The default accepts any password. Once the user and password have been confirmed, the `ftpState` variable can be set to 1.

`: (FTPuser)  \ --`
The default action performed for the USER command. All users are accepted.

`: (FTPpass)  \ --`
The default action performed for the PASS command. All passwords are accepted.

`: (FTPacct)  \ --`
The default action performed for the ACCT command. All accounts are accepted.

33.10 Implemented FTP commands

FTP commands all start with a command name ("verb" in FTP parlance). Each is implemented as a Forth word in the `FTPvoc` vocabulary.

`vocabulary FTPvoc  \ --`
Vocabulary holding FTP commands for execution.

`' FTPvoc >body @ constant FTPwid  \ -- x`
Wordlist holding FTP commands for execution.

`: FTPinterpret  \ --`
Interpret the current line containing an FTP command.

`also FTPvoc definitions`
Start of FTP command definitions.

`: USER  \ -- ; USER <name>`
Handle the USER command.

`: PASS  \ -- ; PASS <password>`
Handle the PASS command.

`: ACCT  \ -- ; ACCT <password>`
Handle the ACCT command.

`: SYST  \ -- ; SYST`
Handle the SYST command.

`: STAT  \ -- ; STAT`
Process the STAT command. Commands with parameters are rejected.

`: HELP  \ -- ; HELP [<param]`
The HELP command just describes the system.

`: NOOP  \ -- ; NOOP`
The NOOP command just returns good.

`: STRU  \ -- ; STRU F`

Obsolete command - we just accept F.

`: MODE \ -- ; MODE S`
Obsolete command - we just accept S.

`: TYPE \ -- ; TYPE <params>`
Process the TYPE command. Valid parameters are

```
 A, A N, I, L 8
```

`: QUIT \ -- ; QUIT`
Process the QUIT command.

`: ABOR \ -- ; ABOR`
Process the ABORt command.

`: PORT \ -- ; PORT a1,a2,a3,a4,ph,pl`
Process the PORT command. The IP address and port are saved in `my_ftpDataIP` and `my_ftpDataPort`.

`: PASV \ -- ; PASV`
Process the PASV command. RFC 0959 does not document PASV well. See `http://cr.yp.to/ftp/retr.html` for more details. A successful response is of the form:

```
  227 Entering Passive Mode (a1,a2,a3,a4,ph,pl)
```

`: RETR \ -- ; RETR <filepathname>`
Process the RETRieve command. Starts transmission of a server file to the FTP client.

`: CWD \ -- ; CWD dirpath`
Process the CWD command.

`: XCWD CWD ; \ -- ; XCWD dirpath`
Process the XCWD command.

`: PWD \ -- ; CWD dirpath`
Process the PWD command.

`: XPWD PWD ; \ -- ; XCWD dirpath`
Process the XPWD command.

`: LIST \ -- ; LIST <filespec/dirspec>`
Process the LIST command. The parameter is optional. If there is no parameter, the current directory is listed. If the parameter is a file, the details of that file are listed. If the parameter is a directory, the files and directories are listed in "sort of" the Unix *ls* format.

`: NLST \ -- ; LIST <filespec/dirspec>`
Process the LIST command. The parameter is optional. If there is no parameter, the current directory is listed. If the parameter is a directory, only the files are listed by returning one name per line.

`: RNFR \ -- ; RNFR <filespec>`
Process the RNFR command, checking that the file exists.

`: RNTO \ -- ; RNTO <filespec>`
Process RNTO, the second part of the file rename operation.

`: DELE \ -- ; DELE <filespec>`
Process the file delete command.

`: MKD \ -- ; MKD <pathspec>`

Process the make directory command.

`: RMD \ -- ; RMD <pathspec>`
Process the delete directory command.

`previous definitions`
End of FTP command definitions.

33.11 FTP service tasks

`: FTPcommands \ --`
Empty the return stack, store 0 in `SOURCE-ID`, and enter interpretation state. `FTPcommands` repeatedly inputs an command line and `FTPinterpret`s it. Note that any task that uses `FTPcommands` must initialise `'TIB`, `BASE`, `IPVEC`, and `OPVEC`.

`: FTPservice \ --`
The FTP service task launched for each established Telnet connection.

`: #FTPconns \ -- u`
Return the number of FTP connections. This is the number FTP service tasks running. Each one needs two sockets.

33.12 FTP listening task

`: FTPserverPass \ --`
One iteration through the FTP server.

`: FTPserver \ -- ; stay here forever`
The FTP listening task.

`0 value FTPtask \ -- 0|task`
Returns 0 or the FTP server task if running.

`: startFTPserver \ --`
Start the FTP server task.

`: stopFTPserver \ --`
Stop the FTP server.

33.13 Diagnostics

This code is only compiled if the `EQU`ate `DIAGS?` is set non-zero.

`: +ftp initfatfs . powernet ;`
Start the file system and PowerNet.

`: .FTPchain \ --`
Display data about the FTP sockets.

34 WWW Support

The source code for these string and time functions is in *PNmisc.fth*.

34.1 Strings

`: -leading \ c-addr u -- c-addr u`

Modify a string address/length pair to ignore leading spaces.

`: addchar \ char string --`

Add the character to the end of the counted string

`: append \ c-addr u $dest --`

Add the string described by C-ADDR U to the counted string at $DEST.

`: is= \ c-addr1 c-addr2 u -- flag`

Compare two same-length strings/memory blocks, returning TRUE if they are identical. The comparison is case insensitive.

`: str= \ addr1 len1 addr2 len2 -- flag`

Compare two addr/len memory blocks, returning TRUE if they are identical both in length and contents. The comparison is case sensitive.

`: ISEARCH \ c-addr1 u1 c-addr2 u2 -- c-addr3 u3 f`

Search the string *c-addr1/u1* for the string (\i{c-addr2/u2}. If a match is found return *c-addr3/u3*, the address of the start of the match and the number of characters remaining in *c-addr1/u1*, plus flag *f* set to true. If no match was found return *c-addr1/u1* and *f=0*. Case insensitive for English.

`: left-string \ caddr len char -- caddr len'`

Given a string and a delimiter, return the string to the left of the delimiter.

`: right-string \ caddr len char -- caddr len'`

Given a string and a delimiter, return the string to the right of the delimiter.

`: extract-string \ caddr len char -- caddr' len'`

Given a string, return the substring between the delimiters, ignoring leading delimiters.

`: jump-string \ caddr len char -- caddr' len'`

Ignore leading delimiters, and return the string to the right of the next delimiter character including the delimiter.

`: split \ addr len char -- laddr llen raddr rlen`

Extract a substring at the start of addr/len, returning *raddr/rlen* the string remaining after char and the substring *laddr/llen* which includes all characters before but not including *char*. If the string does not contain the character, *raddr* is *addr*+*len* and *rlen*=0.

`: split$LR \ addr len char -- laddr llen raddr rlen`

Extract a substring at the start of *addr/len*, returning *raddr/rlen* the string remaining after char and the substring *laddr/llen* which includes all characters before but not including *char*. If the string does not contain the character, *raddr* is *addr*+*len* and *rlen*=0.

`: split$RL \ addr len char -- raddr rlen laddr llen`

As `split$LR` but the left string is returned topmost.

`: csplit \ addr len char -- raddr rlen laddr llen`

Extract a substring at the start of addr/len, returning the string raddr/rlen which includes char (if found) and the string laddr/llen which contains the text to left of char. If the string does not contain the character, raddr is addr+len and rlen=0.

`: string>n \ addr len radix -- value`
Converts an ASCII string to a number in the given base. Characters are converted to upper case before conversion.

`: string>dec \ caddr len -- n`
Converts an ASCII string as a decimal number.

`: string>hex \ caddr len -- n`
Converts an ASCII string as a hexadecimal number.

`: dec>string \ n -- caddr len`
N is converted to a signed decimal string.

`: hex>string \ n -- caddr len`
N is converted to an unsigned hexadecimal string with a leading $ character

`: is= \ c-addr1 c-addr2 u -- flag`
Compare two same-length strings/memory blocks, returning TRUE if they are identical. The comparison is case insensitive.

34.2 Time and date

These functions rely on the ANS Forth word `TIME&DATE ( -- s m h dd mm yyyy )` and the non-standard `DOW ( -- dow, 0=Sun)` to get the day of the week.

`create days$ \ -- addr`
String containing 3 character text for the days of the week.

`create months \ -- addr`
String containing 3 character text for the months.

`: .dow \ --`
Display day of week.

`: .2r \ n --`
Display n as a two digit number with leading zeros.

`: .4r \ n --`
Display n as a four digit number with leading zeros.

`: .Time&Date \ --`
Display the system time The format is:

```
hh:mm:ss dd Mmm yyyy
```

`: .AnsiDate \ zone --`
Display the day of week, date and time. If zone is 0 GMT is displayed. The format is:

```
dow, hh:mm:ss dd Mmm yyyy [GMT]
```

`: IncludeMem \ caddr len --`
`INCLUDE` a block of memory as if it were a file.

34.3 Test code

35 HTTP Server

The Powernet HTTP server is a multi-threaded server that can accept multiple connections limited only by available heap space. For details of the server architecture see *Servers.fth*.

Web pages may be served from memory or from the FAT file system. If you are using the FAT file system, you can configure the root directory for web pages and the name of the home page. If you do not specify them, they will default to *\PAGES* and */home.htm*.

```
create pagedir$  ", \HTTP"      \ -- caddr
\ *G The base directory for pages as a counted string.
\ ** The directory name must not end in a separator.
create HomePage$  ", /home.asp" \ -- caddr
\ *G Holds the counted string for the home page.
```

35.1 HTTP specific data

These data definitions are required by each HTTP task. The data is allocated at the start of the task and released when the task is `TERMINATE`d. The chain `SVCHAIN` links all the service tasks.

For details of `QVAR`s see the CGI section below.

`#80 equ HTTPPort#  \ -- n ; standard is 80`
Define the port used for the HTTP server. The standard port is 80. Moved to *PNconfig.fth*.

`#8 equ #QVARS  \ -- n`
Number of `QVAR`s in each connection and the common area.

`#20 equ /QvarName  \ -- n`
Length including count byte of a `QVAR`'s name.

`#20 equ /QvarData  \ -- n`
Length including count byte of a `QVAR`'s data area.

`struct /QVarRec  \ -- n`
Structure of a single `QVAR`.

`#QVARS /QVarRec * equ /QVARS  \ -- n`
Size of a QVAR buffer.

`struct /HTTPdata  \ -- n`
The standard service data structure is extended for HTTP.

`0 equ NO_SCRIPT  \ -- 0`
No selected script language identifier.

`-1 equ FORTH_SCRIPT  \ -- -1`
Selected script language identifier for Forth.

`$80000000 equ WEBS_CLOSE  \ -- mask`
Status bit to close the connection.

`$40000000 equ WEBS_SCRIPTERR  \ -- mask`
Status bit for a script error.

```
$00000100 equ WEBS_ASP          \ -- mask
```
Status bit for ASP processing.

```
$00000002 equ WEBS_KEEP_ALIVE
```
Status bit for "keep alive" handling.

```
0 equ PLAIN_TYPE        \ -- 0
```
The data type indicator for PLAIN data.

```
1 equ GIF_TYPE          \ -- 1
```
The data type indicator for GIF data.

```
2 equ HTML_TYPE         \ -- 2
```
The data type indicator for HTML data.

```
3 equ ASP_TYPE          \ -- 3
```
The data type indicator for ASP data - HTML data with server side scripting. Output **is not** buffered.

```
4 equ JPEG_TYPE         \ -- 4
```
The data type indicator for JPEG data.

```
5 equ XML_TYPE          \ -- 5
```
The data type indicator for XML data.

```
6 equ ASMX_TYPE         \ -- 6
```
The data type indicator for ASMX data in web services - XML data with server side scripting.

```
7 equ ASPX_TYPE         \ -- 7
```
The data type indicator for ASP data - XML data with server side scripting. Output **is** buffered.

```
8 equ CSS_TYPE          \ -- 8
```
The data type indicator for CSS data.

```
: WebErrCode    \ -- addr
```
Return the address of the service's error code.

```
: WebStatus     \ -- addr
```
Return the address of the service's status flags.

```
: WebVars       \ -- addr
```
Return the address of the service's connection variables.

```
: WebScriptId   \ -- addr
```
Return the address of the service's script identifier.

```
: WebOpFlags    \ -- addr
```
Return the address of the service's operation flags.

`bit0` set if primary headers have been read.

```
: WebPageDef    \ -- struct
```
Return the address of the file page definition structure. This word is forward referenced.

```
: TestWebFlag   \ mask -- n|0
```
Test the mask bits in the service's status flag cell.

```
: SetWebFlag    \ mask --
```
Set the mask bits in the service's status flag cell.

```
: ClrWebFlag    \ mask --
```

Clear the mask bits in the service's status flag cell.

```
: SetHttpTimeout        \ time --
```
Set the timeout target for HTTP. A value of 0 indicates no timing.

```
: GetHttpTimeout        \ -- time
```
Get the timeout target for HTTP. A value of 0 indicates no timing.

35.2 HTTP vectored I/O

35.2.1 Stream socket

The HTTP server establishes its own generic I/O based on that in *SERVERS.FTH* in order to handle special character processing in the future.

```
create ConsoleHTTP      \ -- addr ; OUT managed by upper driver
```
Function despatch table for HTTP I/O. OUT is managed by the upper level driver.

```
: HTTPio        \ --
```
Select HTTP as the console.

```
: Init-ConsoleHTTP      \ --
```
Initialise for console I/O by HTTP. Note that the HTTP must have been set up and and the private service area initialised.

35.2.2 Output to a memory buffer

For correct handling of error responses, some browsers require the HTTP "Content-Length" field to be defined. This means that the output length must be known before sending. Consequently, error messages are buffered before transmission.

N.B. All output to the buffer is unchecked for overflow. Checking is the responsibility of the application.

```
cell +user my_opbuff    \ -- addr
```
Holds the current output buffer, the first cell holding the buffer length.

```
: BuffEmit      \ char --
```
Send a character to the buffer.

```
: BuffType      \ caddr len --
```
Send a string to the buffer.

```
: BuffCr        \ --
```
Send a CR/LF pair to the buffer.

```
create ConsoleBuff      \ -- addr ; OUT managed by upper driver
```
Function despatch table for HTTP buffered I/O. OUT is managed by the upper level driver.

```
: Buff$         \ -- addr len
```
Return the address and length of the HTTP output buffer.

```
: Init-ConsoleBuff      \ len -- addr|0
```
Initialise for console I/O by the HTTP output buffer. *Len* is the required size of the buffer and the address of the buffer is returned for success, or zero is returned if the buffer could not be allocated. The first cell of the buffer contains the length used, the rest is for data.

```
: Term-ConsoleBuff      \ --
```
Terminate buffer I/O by freeing the buffer.

35.3 Diagnostic control

```
1 value httpDiags?      \ -- n
```
Set this non-zero to get diagnostic information.

```
: [hd   httpDiags? if consoleio decimal  ;
```
A `COMPILER` macro used to surround debug code, and terminated by `HD]`.

```
[HD  ." debug message"  HD]
```

```
: hd]   HttpIo  endif  ;
```
Terminates a `[HD ... HD]` structure.

```
: .hdLine       \ caddr len --
```
Display text with leading CR on Forth console. If `httpDiags?` is set to zero, no action is taken.

35.4 Transmit Utilities

```
PDATA_MAX equ WEB_SIZE  \ -- n
```
Maximum web content sent in one packet.

```
: WebSend       \ caddr len --
```
Send an arbitrary sized data block to the HTTP client.

35.5 CGI Support

Common Gateway Interface (CGI) defines how data is passed between a browser (client) and a server. The equivalent of a variable is a name/value pair (e.g. submit=send). Such pairs are sent by the browser after filling in a form.

CGI variables are "URLencoded" as key/value pairs of the form:

```
key1=value1&key2=value2&...
```

When a form response is sent with a GET message, the CGI variables are sent after the URI separated from it by a '?' character. When a form response is sent with a POST message, the CGI variables appear in the body of the message.

When a GET message is used, PowerNet receives the CGI variables before it knows what to do with them. They are saved in structures called `QVAR`s, which hold counted strings. Numeric values are converted from text by the appropriate Forth words. Name comparision is case-insensitive.

PowerNet manages two sets of these variables. The first is a common set, which may be used to hold system data such as the host's IP address. The second set is allocated per connection as part of the service data, and only exists for the duration of the connection.

```
#20 equ /QvarName       \ -- n
```
Length including count byte of a `QVAR`'s name.

```
#20 equ /QvarData       \ -- n
```
Length including count byte of a `QVAR`'s data area.

```
struct /QVarRec         \ -- n
```
Structure of a single `QVAR`.

```
#32 equ #QVARS  \ -- n
```
Number of QVARs in each connection and the common area.

```
#QVARS /QVarRec * equ /QVARS    \ -- n
```
Size of a QVAR buffer.

```
/QVARS buffer: commonQvars      \ -- addr
```
The buffer area for the common QVARs.

```
$80000000 equ NO_VAR_SET        \ -- n
```
Indicator returned when a QVAR has not been set.

```
/QvarName 1 chars - equ /QV.name        \ -- n
```
Maximum length of a QVAR's name.

```
/QvarData 1 chars - equ /QV.data        \ -- n
```
Maximum length of a QVAR's string data.

```
: (.Qvars)      \ addr --
```
Display the variables in the given table.

```
: .Qvars                \ --
```
Display the common and connection variables.

```
: (freeqvar)    \ table -- addr|0
```
Find a free QVAR in the given table, and return its address or zero if no free space is available.

```
: freeQvar      \ -- addr|0
```
Find a free QVAR for the current connection, and return its address or zero if no free space is available.

```
: freeCommonQvar        \ -- addr|0
```
Find a free QVAR in the common QVARs, and return its address or zero if no free space is available.

```
: (findQvar)    \ caddr len table -- addr|0
```
Try to find a QVAR in the given table. Case insensitive.

```
: findQvar      \ caddr len -- addr|0
```
Try to find the given QVAR name, returning the address if found or zero if not found. Case insensitive.

```
: %xx>char      \ caddr len -- caddr' len' char
```
Convert the three character sequence "%xy" as a hexadecimal two-digit number. Step over the string.

```
: decodeURL$    \ caddr len dest dlen --
```
Converts the source string *caddr/len* from a URL-encoded string to a decoded counted string in the buffer *dest/dlen*. No error checking is performed. Decoding converts '+' characters to a space and "%ab" ('%' follwed by two hex digits) sequences to their character codes.

```
: setQvarData   \ caddr len qvar --
```
Use the given URL encoded string *\i{caddr/len) to set the data area of the given *qvar*.

```
: setQstring    \ name nlen string slen --
```
Set the connection QVAR *name/nlen* to contain *string/slen*. If the name already exists in the connection or common QVARs it is overwritten. If the QVAR does not exist it is created in the connection's QVAR set. If there is no space for it, the request is ignored.

```
: setCommonQstring      \ name nlen string slen --
```

Set the common QVAR *name/nlen* to contain *string/slen*. If the QVAR does not exist it is created. If there is no space for it, the request is ignored.

```
: GetQstring    \ name nlen -- caddr len
```
Return the text for a string. If the variable cannot be found, "???" is returned.

```
: .Qstring      \ name nlen --
```
Output the text for a QVAR using GetQstring above.

```
: websetvars    \ caddr len --
```
Process a query string. A query string has the form:

```
name=value&name=value...
```

WEBSETVARS can be used with any query string.

```
: WebQueryVars  \ caddr len --
```
This is used by GET with query packets.

```
: init-CommonQvars      \ --
```
Initialise the common QVARs.

```
: init-WebVars  \ --
```
Initialise the connection QVARs.

35.5.1 Numeric QVARS

```
: setQvar       \ caddr len n --
```
Set the given QVAR to *n*, which is held as a signed decimal string.

```
: getQvar       \ caddr len -- n
```
Return the value held in the QVAR as a signed decimal number. If the QVAR does not exist NO_VAR_SET is returned. If the string cannot be converted, zero is returned.
CEM specific numeric items are commented out.

```
: QvarString?   \ caddr len -- type
```
Return true if the variable is a string variable.

35.6 ASP Support

ASP stands for "Active Server Pages". In PowerNet, these are HTML pages which are modified by PowerNet when served. See *Examples\PowerNet\TestPages\thanks.asp* for an example.

The script language is Forth itself. Inside an HTML document, scripting commands (Forth source) are contained inside tags of the form:

```
<% Forth_code %>
```

It is important that the script delimiters <% and %> are surrounded by white space, otherwise the very simple parser will fail. Before any scripting can be performed, the first command must be on one line:

```
<% language=forthscript %>
```

After that, Forth source code can be interpreted. Note that any CGI variables (QVARS above) are available. For example, if a form was submitted with a GET request:

```
GET form1.asp?sname=Robert&send=submit
```

You can display the data using a script such as:

```
<% s" sname" .qstring %>
```

Pages can be served from a linear memory image, or from the FAT file system. When scripts are served from memory, each script section **must** be on a single line. When serving scripts from files, the script section can extend over several lines.

`: script_code  \ -- caddr len`
Returns the command to select Forth as the scripting language.

`: asp_header$  \ -- caddr len`
ASP command header string.

`: asp_tail$  \ -- caddr len`
ASP command tail string.

`: ScriptEngine  \ caddr len --`
Processes the string as Forth source. The data stack is checked on return to ensure system integrity.

`: AspProcess  \ caddr len --`
If Forth has been selected as the script language, pass the string to `SCRIPTENGINE`, otherwise try to find the Forth script command. In practice, this means that the language selection command must be on its own as the first script section.

`: AspRequest  \ caddr len --`
Extract script section and try to process it. All output is done using `TYPE` and `WebSend`.

35.7 Header scanning

HTTP headers are processed by building a list of actions and strings. The action is an xt and the string is theheader text that we are interested in. Each action has the stack effect

```
caddr len --
```

where *caddr/len* is the string after the header and the trailing colon.

```
create UploadHdrs       \ -- addr
  ' doCLength ,  ," Content-Length" align
  ' doCType1 ,   ," Content-Type" align
  0 , 0 ,
```

`: CheckName  \ src slen name nlen -- flag`
Return true if the start of the string *src/slen* contains the entire name *name/nlen.*

`: CheckHeader  \ src slen name nlen -- flag`
Return true if the start of the string *src/slen* contains the entire name *name/nlen* and a trailing semi-colon.

`: doHeader  \ caddr len list --`
Given a string, check it against the given list of headers.

`: ProcessHeaders  \ list --`

We have already received the GET/POST line. Process each header line and finish at the first blank line, ready for the message body. This can be used for the header blocks in forms as well as the for the first header.

`: doCLength \ caddr len --`
Process the string to extract the content length data, a decimal number. A valid results is saved in the HTTP service data's `httpClength` field.

`variable FormType \ -- addr`
Holds the status extracted from the first `Content-Type` header. This set is true if the header includes "multipart/form-data".

`#80 buffer: Boundary$ \ -- addr`
Holds the boundary string.

`: nextChar \ caddr len -- caddr' len' char`
Get the next character from the string and step on.

`: ?MultiPart \ caddr len --`
Check for the *multipart/form-data* field.

`: ExtractValue \ caddr len dest --`
Given a string starting after the name portion of a name/value pair, e.g. *name="text"*, save the value text without any quote marks at *dest*. If there is no value text, the destination is left unchanged.

`: ?Boundary \ caddr len --`
Check for the *boundary=xxx* field.

`: doCType1 \ caddr len --`
Process the string to extract the content type data, which is the multipart form item and the boundary string.

`create BaseHdrs \ -- addr`
Contains a list of the basic header fields to process.

`BaseHdrs value DefHdrs \ -- addr`
Holds the default header list for file and memory pages.

`: ReadHeaders \ --`
Process the headers after the HTTP command line up to the first blank line. **Do not** use this word for reading part headers. If the response headers have already been read, this word is a no-op.

35.8 Form body processing

Words are provided for use with form results submitted as a POST request to an ASP page. You can use these with simple URL encoded or multipart forms. An example of multipart form handling is the file upload application in *Examples\WebPost.fth*. This example includes boundary handling and parsing code.

35.8.1 Tools

Correct operation of this code requires a Content-Length header in the request. In many cases, the body of a POST request is a very long line. Because this server is designed for systems with limited RAM, key/value pairs are `not` read as a single line, but are read and parsed character

by character. Keys are limited to 31 characters, and values to `/SVtib - 32` characters. The service's `TIB` is where they are buffered.

`: BodyLeft \ -- addr`

Returns the address holding the remaining size of the HTTP body. If no Content-Length field has been found, `ReadHeaders` will have set this to -1.

`: -BodyLeft \ n --`

Reduce the remaining content by *n*.

`: CGIname$ \ -- caddr`

Buffer for name assembly.

`: CGIval$ \ -- caddr`

Buffer for data assembly.

`: CGIend? \ -- flag`

Return true if the body content is exhausted.

`: CGIkey \ -- char`

Read the next character. If the content is exhausted or there has been a socket error, LF is returned.

`: ReadKey \ -- ior`

Read the key portion of a key/value pair, returning zero for success.

`: ReadValue \ -- ior`

Read the value portion of a key/value pair, returning zero for success.

35.8.2 Application words

These words will mostly be used inside ASP scripts. Note that the key and value strings are not URL decoded. You can use `decodeURL$` to perform the decode and copy in one operation.

It is assumed that you control both the RAM usage of the PowerNet server and that you control the scripts it serves. We are not trying to reimplement the Apache server!

To read the pairs, call `ReadNextPair` and inspect the return result. If it is good, examine and process the `PairName` and `PairValue` strings. Because you control the scripts, it is sensible to use key names that do not require URL decoding. Especially for user input, some key values may need decoding, e.g. email addresses. This decision is left up to you.

`: ReadNextPair \ -- ior`

Read the next key/value pair and return 0 on success. -1 is returned for an unexpected end of input or socket error, and -2 is returned if input was truncated before the key terminator.

`: PairName \ -- caddr len`

Return the last key/value pair's key text.

`: PairValue \ -- caddr len`

Return the last key/value pair's value text.

`: DumpPairs \ --`

A diagnostic tool for use in scripts. Read and dump all the key/value pairs without preserving them.

35.9 HTTP headers and responses

This section deals with extracting the HTTP command data and with pages served from memory.

`: web-file-name \ caddr len -- caddr len'`
Return the string up to the next '?' character. Then, if the file name starts "HTTP://.../" step over "HTTP://...". This situation can occur with proxy servers.

`: test-web-type \ caddr len -- type`
Return the type code associated with the file extension, e.g ".htm" or ".asp".

`: .HTTP#  \ n --`
Output the HTTP string and code line.

`: .HTTPserver  \ --`
Output the HTTP server name line.

`: .HTTPdate  \ --`
Output an HTTP date line.

`: .HTTPar-none  \ --`
Output the "accept-ranges: none" line.

`: .HTTPcontent  \ type --`
Output the text string for the curent type. Unknown types generate "text/plain".

`: .close/keep  \ --`
Output the default connection type.

`: WebResponse  \ datalen type code --`
Output a suitable HTTP header for this type of data.

`create Null$  \ -- addr`
A null string which may be used as a counted string or a zero-terminated string.

`: .ErrHead  \ len err# --`
Output the error header. If len is non-zero, it is used for a "Content-Length" field.

`: .ErrBody  \ caddr1 len1 caddr2 len2 err# --`
Output a web error body using the given strings and error number. The string caddr2/len2 contains the error description, e.g. "Page not found". If len2 is zero no error text body is sent and caddr1/len1 is discarded. The string caddr1/len1 is displayed if len1 is non-zero, and is usually the resource name that caused the problem.

`: weberror  \ caddr1 len1 caddr2 len2 err# --`
Display a web error message using the given strings and error number. The string caddr2/len2 contains the error description, e.g. "Page not found". If len2 is zero no error text body is sent and caddr1/len1 is discarded. The string caddr1/len1 is displayed if len1 is non-zero, and is usually the resource name that caused the problem.

`create HomePage$  ", /home.htm"`
If not already defined, `HomePage$` is set to contain the counted string */home.htm*.

`: CheckHomePage \ caddr len -- caddr' len'`
Check for a home page string of the forms "/ " or "/" and if found replace with "/home.htm".

`: ServeSMem  \ caddr len type --`
Serve a page from memory, according to its page type extracted from the page name.

35.10 Serving files

`#256 equ /FileUnit \ -- len`
The size of a part of a file processed at once.

`: FileQuery \ -- ; fetch line into TIB`
Reset the input source specification to the console and accept a line of text into the input buffer.

`: AspInterpret \ --`
Process the current input line as if it is text entered at the keyboard.

`: InterpScript \ --`
Interpret a section of ForthScript which may extend over several lines.

`: sendPrev \ addr --`
Send the source line before this address.

`semaphore InterpSem \ -- addr`
Exclusive access semaphore for the Forth interpreter.

`: FileScript \ --`
Read and process a file with server side scripting until EOF or an error. The file is already open and is not closed.

`: AspFileReq \ len --`
Read and process a file of length *len* with server side scripting. The file is already open. All input is done using the `FileCon` Generic I/O device (see *FATcore.fth*. All output is done using `TYPE` and `WebSend`.

`: badFS \ x y z --`
Discard three items and set error flags.

`: (FileSend) \ len buff --`
Serve the file using the given buffer.

`: FileSend \ len --`
Serve a plain file. without scripting. The file is open and the details are in the given `/PageDef` structure.

`: ServeFile \ len type --`
Serve a page from a file, according to its page type extracted from the page name. The request headers are read if they have not already been read.

35.11 HTTP service task

`: ?CloseHTTP \ --`
Run after processing input to see if the connection should be closed.

`: Serve404 \ qaddr qlen --`
Serve a "not found" page.

`: ServePage \ qaddr qlen --`
Serve a page defined by the string *qaddr/qlen* which may include CGI vars. The string is the command line with the command removed.

`: http-cmd \ caddr len -- caddr' len'`
Deal with the command line after the command has been recognised. The input is the complete line. The output is the line without the command.

`: http-get \ caddr len --`

Process a GET command. The input string is the complete input line containing the GET command.

`semaphore PostSem \ -- addr`
Controls access to the single POST handler. POST requests are serialised because they may change the state of the system.

`: http-post \ caddr len --`
Process a POST command. The input string is the complete input line containing the POST command.

`: ParseHTTP \ caddr len --`
Process received HTTP command line. At present we only deal with GET and POST commands.

`: doHTTPinput \ --`
Process any pending input.

`: cleanHTTP \ *sv -- *sv`
Cleans up the HTTP system when a task is shut down from the kill chain.

`: HTTPService \ --`
The HTTP service action or task launched for each established HTTP connection.

35.12 HTTP listening task

Listening tasks or actions are spawned when the HTTP server gets a connection.

`: HTTPServer \ -- ; stay here forever`
The HTTP listening task.

`0 value HTTPtask \ -- 0|task`
Returns 0 or the HTTP server task if running.

`: RunHTTPtask \ --`
Start the HTTP server task.

`: StopHTTPtask \ --`
Stop the HTTP server.

35.13 Notes on memory usage

The majority of page requests are made using a GET request. For these, only the first line of the header needs to be scanned and the request is contained in a single packet whose maximum size is defined in lower layers of the Powernet system. Consequently a fixed size packet buffer is used for HTTP input.

When handling POST requests, e.g. for Web Services, the input data is not in the headers, but is contained in the body of the message. The size of this body is defined by the *Content-Length* header. When handling POST messages, the whole of the incoming message must be read, the body extracted and passed to the message handler. In addition, some handlers may need to process the message header. For example, Web Services need to use the *SOAPaction* header before SOAP version 1.2.

Scripting for output messages introduces the problem that the size of the message body is not (in general) known until the script output has been generated. This means that the *Content-Length* header (sent before the body) cannot be formed until the body has been generated.

To avoid the memory overhead of buffering script output, for ASP file requests no *Content-Length* header is generated and the connection is closed after the response has been sent to indicate that the message is complete. For web services, ASPX pages are served and are buffered, and a *Content-Length* header is generated.

Error messages such as "404 Not found" are always buffered to produce a valid *Content-Length* header because some browsers require this header for error messages.

The *Content-Length* header is not always required for HTTP version 1.1 and above. However, if HTTP 1.0 clients have to be supported the *Content-Length* header must be provided. In this case all scripted operations must be buffered. See RFC2616 for more detils.

35.14 Authentication

Before PowerNet v4.4, authentication was provided by a deferred word. Nobody reported using it, so it has been removed to save memory. This section shows how to put it back if required. All the code to be added can be found towards the end of *HTTP.FTH*.

More flexibility is provided in v4.4 because it is much easier to parse headers (see *Examples\WebPost.fth*), and scripting provides more choice as to which files are secured and which are public.

Add the line below to the `/HTTPdata` structure.

```
int httpLogin                    \ login value
```

Add the following definitions.

`: WebLogin      \ -- addr`
Return the address of the service's login result.

`defer WebAuthenticate   \ caddr len -- res|0`
Given a GET string, returns a non-zero code for permission to carry on. If permission is refused, zero is returned.

`: (WebAuthenticate)     \ caddr len -- res|0`
The default action of `WebAuthenticate` always returns true.
Restore the following code to `HTTP-CMD`.

```
2dup WebAuthenticate ?dup 0= if
  2drop  Null$ 0 s" Invalid login " #401 weberror  exit
endif
WebLogin !                          \ stash good login result
```

`create BaseHdrs \ -- addr`
Contains a list of the basic header fields to process.

36 Web page handling

Examples are given here for pages in memory and pages in files. Feel free to experiment with mixed operation, but note that the code in <*PNet*>*Services**Pages.fth* is maintained by MPE and may/will change with future releases, so we suggest that you keep changes in a separate file.

36.1 Configuration

The following two equates are only used if they are not already defined.

```
1 equ MemPages? \ -- n
```

Set this non-zero to compile the memory pages code.

```
1 equ FilePages?        \ -- n
```

Set this non-zero to compile the file pages code.

36.2 Data structures

Since Web pages may be stored in a file system, other mass storage or memory, a common data structure is used based on the needs of pages stored in CPU memory. Pages stored in CPU memory are identified by name in the `HTTPPAGES.VOC` vocabulary. Executing the word returns the address of the data structure and its type.

```
struct /PageDef \ -- n
```

The structure used for a page definition.
The following equates define the types of pages that are available. They are stored in the `PG.TYPE` field of a `/PageDef` structure.

```
0 equ NoPage    \ -- 0
```

Page not found or unusable.

```
1 equ FilePage  \ -- n
```

The page is in a file.

```
2 equ SMemPage  \ -- n
```

The page is in memory (e.g. Flash) and need not be released.

```
3 equ xtPage    \ -- n
```

The page is created by executing a Forth word, whose xt is in the `PG.addr` field.

```
4 equ HMemPage  \ -- n
```

The page has been loaded into the heap.

```
vocabulary HTTPpages.voc        \ --
```

The default vocabulary used to hold page data.

```
: voc>wid       \ xt(voc) -- wid
```

Return the WID of a vocabulary whose XT is supplied. This definition is implementation dependent.

```
: FindVocPage   \ caddr len -- struct|0 type|0
```

Find a page from its name *caddr/len*, returning the data structure address *struct* and a type code *type*. If the page cannot be found, both return values are zero. The action is to look up the name in the `HTTPpages.voc` vocabulary and extract the page type from the `/PAGEDEF` structure.

36.3 Executable pages

`: xtPage:  \ xt "<name> -- ; -- struct type`
Creates a page which later executes the given action, e.g.

```
' doNewApp xtPage: /NewApp.asp
```

The specified action must have the stack effect

```
qaddr qlen --
```

where the input is the rest of the line after GET or POST.

36.4 Memory pages

`: MemPage:  \ "<Forthname>" "<filename>" -- ; -- struct type`
An `INTERPRETER` definition that creates `<Forthname>` in the `HTTPPAGES.VOC` vocabulary and a `/PAGEDEF` structure, and then loads file <*filename*> into the dictionary as data. At run-time the address of the `/PAGEDEF` structure is returned. When the HTTP server is running it will serve the data in response to a GET request for Forthname.

```
MemPage: /home.htm %IPSTACK%\powernet.htm
```

36.4.1 Example memory pages

These files are compiled by default unless the equate `TestPages?` exists and is set to zero.

`MemPage: /home.htm %IPSTACK%\TestPages\home.htm`
Creates embedded memory page */HOME.HTM* from the data in *%IPSTACK%\home.htm*. The leading '/' is required as it is not removed by the command parser.

`MemPage: /bgimage.jpg %IPSTACK%\TestPages\bgimage.jpg`
The background image for *POWERNET.HTM*.

`MemPage: /favicon.ico %IPSTACK%\TestPages\favicon.ico`
The page icon for these pages.

`MemPage: /contact.htm %IPSTACK%\TestPages\contact.htm`
Creates embedded memory page */CONTACT.HTM* from the data in *%IPSTACK%\contact.htm*.

`MemPage: /form1.htm %IPSTACK%\TestPages\form1.htm`
A simple form.

`MemPage: /thanks.htm %IPSTACK%\TestPages\thanks.htm`
The response to *FORM1.HTM*.

`MemPage: /form2.htm %IPSTACK%\TestPages\form2.htm`
A simple form with ASP and GET handling.

`MemPage: /form3.htm %IPSTACK%\TestPages\form3.htm`
A simple form with ASP and POST handling.

`MemPage: /thanks.asp %IPSTACK%\TestPages\thanks.asp`
The scripted response to *FORM2.HTM*.

`MemPage: /thanks3.asp %IPSTACK%\TestPages\thanks3.asp`
The scripted response to *FORM3.HTM*.

36.5 File pages

The defaults for this section assume that the FAT file system in in use.

`create PageDir$ \ -- addr`
The base directory for pages. This will be prepended to the page name given to `SearchPage`. The string is held as a counted string in the buffer. Make sure that your application sets `PageDir$` to point to its own pages. The directory name must not end in a separator. The version here sets the contents to *\PAGES* and is only used if `PageDir$` is undefined here.

`: PrepFilename  \ caddr len --`
Prepare a URL to be a file name. For Windows/DOS file systems, convert '/' characters to '\'. For Unices, do nothing.

`: [io           \ -- ; R: -- ipvec opvec`
Save the current I/O devices on the return stack.

`: io]           \ -- ; R: ipvec opvec --`
Restore the I/O devices from the return stack.

`: FindFilePage  \ caddr len -- struct|0 type|0`
The action of `SearchPage` for pages in files. Find a page into memory from its name *caddr/len*, returning the structure address and length.

36.6 Page look up

`: SearchPage    \ caddr len -- struct|0 type|0`
Find a page from its name *caddr/len*, returning the data structure address *struct* and a type code *type*. If the page cannot be found, both return values are zero. The action is to look up the name in the `HTTPpages.voc` vocabulary and extract the page type from the `/PAGEDEF` structure. If file pages are required, the page directory is searched.

37 SMTP Primitives

This code can send any arbitary text via standard eMail systems using the Simple Mail Transfer protocol (SMTP).

Recommended reading includes:-

`RFC_821` Simple Mail Transfer Protocol

`RFC_822` ARPA Internet Text Messages

```
1 value smtpdiags?      \ -- n
```
Set this non-zero to get diagnostic information.

```
#200 equ SMTPwait       \ -- ms
```
Number of milliseconds to wait for an SMTP response from the server. If you are using a slow connection you may have to increase this value.

```
#256 equ MAXRXSIZE      \ -- n
```
Size of the response buffer for messages from the SMTP server.

```
MAXRXSIZE buffer: SMTPin        \ -- addr
```
The SMTP receive buffer.

```
: SMTP::Wait    \ hsock --
```
Wait for a response from the server for up to SMTPWAIT milliseconds.

```
: SMTP::Error?  \ hsock -- flag
```
Read a response back from the SMTP server via the supplied socket and return a TRUE flag if the server's response indicates an error.

```
: SMTP::Connect \ ipaddr port# -- hsocket|0
```
Attempt to create a socket and connect to an SMTP Server. IPADDR is an ipaddress and PORT# is the requested port. SMTP Servers are almost always found on port 25 (decimal).

```
: SMTP::Disconnect      \ hsock --
```
Disconnect from a SMTP discussion by closing our socket.

```
: (SMTPWrite)   \ hsock c-addr u --
```
Write a buffer out via a socket.

```
: SMTP::Write   \ hsock c-addr u --
```
Write a buffer out via a socket. Used to send text strings to the SMTP Server.

```
create crlf$    \ -- addr
```
A counted string holding a CR/LF pair

```
: (SMTP::WriteLn)       \ hsock c-addr len mode --
```
As 'SMTP::Write' but followed by a CR/LF pair. Mode is a TCP flag which is normally 0, but is set to TCP_PSH by SMTP::WRITEFIELD below.

```
: SMTP::WriteLn \ hsock c-addr u --
```
As 'SMTP::Write' but followed by a CR/LF pair.

```
: SMTP::WriteField      \ hsock field-addr field-len val-addr val-len -- flag
```
Write a 'field' to the SMTP server. In this case a 'field' consists of a 'field name' and a 'text value' followed by an EOL. Any field written prompts the server to make a response which is checked for an error condition to form the return flag.

38 SMTP Demonstration

38.1 Configuration

These definitions should be changed for your system.

```
create SMTPserver       \ -- addr
```
Holds the remote SMTP server IP address in network order. Modify this for your own remote SMTP server.

```
#25 constant SMTPport   \ -- port#
```
The port number used by SMTP servers.

```
: Sender$       \ -- addr len
```
Returns a string containing the sender's email address. Note the use of the angle brackets.

```
: Receiver$     \ -- addr len
```
Returns a string containing the receiver's email address. Note the use of the angle brackets.

```
: Domain$       \ -- addr len
```
Returns a string containing the sender's domain name. Note the use of the angle brackets.

38.2 Sending mail

```
: SendBody      \ hsock --
```
Send the body of the message. All errors are handled by THROWing.

```
: send-demo-email       \ -- ior
```
This shows how to send eMail via SMTP. The ior is returned 0 for success.

39 Ethernet and Internet configuration

The file *WebConfig.fth* defines configuration data and tools for configuring the Ethernet controller, TCP/IP settings and other unit details such a remote web site that your equipment connects to. These tools are useful during both development and production for initial unit setup.

You can use this file as a model for your own hardware. Please copy this file from the installation folder to your application folder before use.

To avoid problems with defaults defined in other files, compile *WebConfig.fth* before the Ethernet driver or other PowerNet code.

```
include %AppDir%\PNconfig           \ PowerNet configuration
include %ExampleDir%\NetCode        \ network order stuff
include %AppDir%\WebConfig          \ configuration tools
include %CpuDir%\drivers\EtherSAM7x \ Ethernet driver
include %IpStack%\PowerNet.bld      \ PowerNet build
```

The following equates select Flash or EEPROM storage for configuration data. Only one should be set. Both assume that the standard MPE tools for EEPROM and Flash read/write are in use. If these tools are not being used you will have to write your own versions of `DataFlash>RAM` and `RAM>DataFlash` below}.

```
1 equ EEPROMconfig?     \ -- x
```
Set non-zero to use EEPROM configuration.

```
0 equ Flashconfig?      \ -- x
```
Set non-zero to use Flash configuration.

39.1 EEPROM/Flash area definition

The structure declared here defines the layout of the configuration data region.

Layout of serial EEPROM or a Flash region which holds configuration information.

```
#1024 constant /CfgData \ -- n
```
Size of the configuration data area.

```
#256 equ /URL   \ -- len
```
Size of a buffer of a domain or page name.

```
[cfg
  4 cfg: Emagic                 \ $5555AAAA
  4 cfg: Eunit#                 \ 4 byte unit number
  4 cfg: EcustID                \ 4 byte customer ID number
  6 cfg: EMACaddress            \ 6 byte Ethernet MAC address
  1 cfg: EDNS?                  \ non-zero if we can do DNS lookups
  1 cfg: EDHCP?                 \ non-zero for DHCP
  4 cfg: EIPaddress             \ 4 byte IPv4 IP addresss, 0/-1 for DHCP
  4 cfg: EEnetIPMask            \ 4 byte network IP mask
```

```
  4 cfg: EIPGateway             \ 4 byte IP gateway
  4 cfg: EPollServer            \ 4 byte Poll server IP address
  4 cfg: EPollPort              \ 4 byte Poll server port
/URL cfg: EPollURL              \ server domain name
/URL cfg: EPollPage             \ server page name
  4 cfg: ETransaction#          \ Transaction number
  0 cfg: Elast                  \ defines size used
cfg]

create CfgTemplate      \ -- addr
```

Default configuration which must match the configuration structure above.

39.2 Runtime data

```
6 buffer: EtherAddress  \ -- addr
```

Holds the Ethernet MAC address (six bytes). Note that you must obtain these from the IEEE (www.ieee.org) or from other sources.

```
1 buffer: DNS?          \ -- addr
```

If non-zero, the first byte indicates that DNS should be used to obtain the poll server's IP address.

```
1 buffer: DHCP?         \ -- addr
```

If non-zero, the first byte indicates that DHCP should be used to obtain the unit's IP address from a local DHCP server

```
4 buffer: IpAddress     \ -- addr
```

Holds the Ethernet IP address (four bytes). The range 192.168.xxx.yyy is commonly used for private networks. The data is in network order.

```
4 buffer: EnetIPMask    \ -- addr
```

IP mask for addresses on the Ethernet port. The data is in network order.

```
4 buffer: IPGateway     \ -- addr
```

Gateway attached to Ethernet port. The data is in network order. Set to 0.0.0.0 if there is no gateway.

```
4 buffer: PollServer    \ -- addr
```

Polling server IP address in network order. This is the server that the reader unit will connect to. The data is in network order.

```
4 buffer: PollPort      \ -- addr
```

Polling server port address, defaults to 80.

```
/URL buffer: PollURL    \ -- addr
```

Buffer for the poll server's domain name.

```
/URL buffer: PollPage   \ -- addr
```

Buffer for the page to be accessed by polling operations.

```
0 value Unit#   \ -- u
```

Unit serial number.

```
0 value CustID# \ -- u
```

Unit customer ID number.

```
0 value Transaction#    \ -- u
```

Transaction number used by the client code.

39.3 Flash and EEPROM routines

Select one set of these routines as required.

39.3.1 Flash

`SecTab dup @ cells + @ FlashBase + constant CfgFlash \ -- addr`
For configurations in Flash, we default to using the last sector of the flash for configuration storage. Systems that use fixed size pages, e.g. Atmel SAM7X CPUs, should predefine `CfgFlash`.

`Elast buffer: CfgBuff \ -- addr`
Buffer used during configuration read and write.

`: SetDefaults \ --`
Flash: Write the default configuration to Flash. Usually run during production configuration before `NetSetup` below.

`: DataFlash>RAM \ --`
Flash: Read the TCP/IP stack and unit settings from the Flash configuration area. Data is copied to RAM if the magic number is correct, otherwise, the default configuration is programmed and used.

`: RAM>DataFlash \ --`
Flash: Write the TCP/IP stack settings to the configuration Flash.

39.3.2 Serial EEPROM

`: DataFlash>RAM \ --`
EEPROM: Read the configuration from EEPROM.

`: RAM>DataFlash \ --`
EEPROM: Write the configuration settings to the EEPROM.

39.4 Set up operations

39.4.1 Displaying and Entering IP addresses

`: (GetIPaddress) \ -- ipaddr|INADDR_NONE|0`
Get an IPv4 address from the user. `INADDR_NONE` (-1) is is returned if the entry is bad, and zero is returned for a zero length entry. Use `.IPv4 ( ipaddr -- )` to display an IP address.

`: GetIPaddress \ caddr -- ior`
Get an IP address from the console in the form `"192.168.0.55"` (base is decimal). A good result is saved at *caddr* and ior is returned 0 for success or a null entry, or other for a bad entry. Note that *caddr* will always be modified.

`: GetIP \ caddr -- ior ; 0=success`
Get an IP address from the console in the form `"192.168.0.55"` (base is decimal). A good result is saved at *caddr*.

39.4.2 Displaying and Entering MAC addresses

`: .MACaddress \ caddr --`
Display the MAC address at *caddr*.

`: getByte \ addr char -- addr'`
Collect an integer delimited by char and store it at addr, returning addr+1.

`: getData \ addr char n --`

Collect n integers delimited by char, and store the bytes sequentially at addr.

`: GetMACaddress \ caddr -- ior ; 0=success`
Get an Ethernet MAC address from the console in the form "aa-bb-cc-dd-ee-ff" (base is hex). The result is saved at *caddr* and ior is returned 0 for success or a null entry, or other for a bad entry. Note that *caddr* will always be modified.

`: GetMac \ caddr -- ; input new MAC address`
Get a new MAC address from the user console.

39.4.3 Setup proper

`: y/n? \ -- flag`
Wait for a key and return true if the key is Y.

`: EtherSetup \ --`
Set up the unit's Ethernet address.

`: getC$ \ caddr --`
Get a counted string of up to `/URL` characters from the keyboard and save it at *caddr*. This version is compiled if a heap is present.

`: getC$ \ caddr --`
Get a counted string of up to 62 characters from the keyboard and save it at *caddr*. This version is compiled if a heap is not present.

`: (getU) \ caddr -- ior`
Get an unsigned number and place it at caddr. Return 0 for success or a null entry.

`: getU \ caddr --`
Get an unsigned number and place it at caddr.

`: .BadE$ \ --`
Warning for unconfigured string.

`: E$. \ caddr --`
Display the counted string unless the count is $00 or $FF, in which case display a warning.

`: NetSetup \ --`
Configure the unit for the network and server.

`: .Config \ --`
Display the Flash configuration area.

`: (UnitSetup) \ --`
Used during production test to set up the unit serial number and customer number.

`: UnitSetup \ --`
Used during production test to set up the unit serial number and customer number.

`: .Config \ --`
Display the EEPROM configuration area.

40 POST handlers and HTTP updates

The file *<Pnet>/Examples/WebPost.fth* provides example handlers for HTTP POST requests. According to the RFCs, GET requests are made when the server is left unchanged by the request. POST requests are made when the state of the server is (or may be) changed by the request.

The example here is of uploading a new application binary image to the server. `DEFER`red words are used to handle the target specific actions. This example can be used as the basis of other POST handlers, especially those using multipart forms. For details of form encoding, see RFCs 2045 and 2046. These can be obtained from `http://www.rfc-editor.org`.

To use the system, you need PowerNet version 4.62 or later. Compile the file *<Pnet>/Examples/WebPost.fth* after the PowerNet build file. When the system is running, point your browser at the page *Reflash.htm*. When you have selected the required file, press the Submit button. This returns the form results and new binary to a page called *NewApp.asp*, which is actually a Forth word that performs the process. When it is complete, the page */naresp.asp* is served to display the results to the user.

To provide flexibility in how the new binary is handled, a simple interface is provided that can be used to save the new binary code to Flash or a file system, e.g. on an SD card.

40.1 Discussion

The requirement is to be able to point any browser at the PowerNet web server, and to be able to upload a new binary image to it using a form for data entry. What is done with this file is application dependent, but it may replace the existing application, or it may be saved to an SD card.

40.1.1 Form

In order to send binary data to a server (file upload), a POST request must be made by the form. Here is the HTML for an example form. Following sections indicate the response, in our case using Firefox.

```
<html>
<head><title>Select new application</title></head>
<body>
<h4>Select application</h4>
<p>Select the application using the Browse button,<br>
then press the "Submit" button to send it to me.<br>
Once you have pressed "Submit" you are committed.<br>
To avoid sending a file, return to the previous page.
</p>
<form enctype="multipart/form-data" method=POST action="newapp.asp">
  <table>
  <tr>
    <td>New application file</td>
    <td colspan=4><input type=file name=appfile></td>
  </tr>
  <tr>
    <td>Upload to remote:</td>
    <td colspan=2><input type=submit name=send value=Submit></td>
  </tr>
  </table>
</form>
</body>
</html>
```

40.1.2 Headers

The following is a response by Firefox to the form above. Note that whenever you press a button of the submit type, you get the complete response, including any selected file.

```
POST /newapp.asp HTTP/1.1
Host: 192.168.0.227
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.14) Gecko/20080404 Fir
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,ima
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://192.168.0.227/Reflash.htm
Content-Type: multipart/form-data; boundary=---------------------------805274455224
Content-Length: 9386

-----------------------------805274455224
```

The `Content-Type` and `Content-Length` headers are essential. The first tells us that this is a form response in the style we need and the second tells us the overal size of the response. If you need security, you can use the `Referer` field to indentify which host and file contained the form.

In particular the `Content-Type` field contains a `boundary` string. This is the only data that identifies where one part of the form starts and ends.

40.2 Form boundaries

The boundary string is defined by the browser. The PowerNet server has **no** control over it. However, it is mandatory that the string is **not** contained in the data.

The example below shows three versions of the boundary string.

```
--------------------------805274455224
----------------------------805274455224
----------------------------805274455224--
```

The first is the separator defined by the `boundary` portion of the `Content-Type` header. Under some circumstances, this may be delimited by '"' characters which are not part of the boundary string.

The second is applied to all part separators except the last one. The second form is the same as the first but with two leading '-' characters. It is always preceded by a CR/LF pair and terminated by a CR/LF pair.

The third marks the end of the form. It is identical to the second type but has two additional '-' characters at the end of the line.

40.2.1 Form data

Using the form above, we receive two blocks of form data between boundary markers.

```
Content-Disposition: form-data; name="appfile"; filename="memcopy.s"
Content-Type: application/octet-stream

/******************** (C) COPYRIGHT
...
```

Note that in our form, the submit button is after the file browser, so the file is sent first! The order in which the elements of the form are sent is the order in which they appear in the HTML of the form.

The `Content-Disposition` header tells us which element of the form is being returned, and the `Content-Type` header tells us the file is being transferred as 8 bit binary. There is blank line (CR/LF pair) between the last header and the start of the binary data.

Note that there is nothing to tell us the size of the data! We `must` rely on detecting the boundary separator unless there is some magic data at the start of the binary data section.

```
Content-Disposition: form-data; name="send"

Submit
```

This represents the Submit button. You can have several buttons of type `submit`. They will

each cause the form response. You can use them to decide where to place the data, e.g. primary or secondary application.

If you want a "Cancel" button, your page will need a second form pointing to a different page.

40.2.2 After the form

After the last item of form data will come the residue, which is usually null. Note that more than one form may be contained in a web page.

40.2.3 Restrictions

The following assumptions and restrictions should be noted.

- Only the last boundary string defined in the message headers is used. Thus nested boundary strings are not supported.
- It is assumed that a boundary string is followed by CR/LF or -/-/CR/LF.
- Very little checking of the message is performed.
- For simplicity, compatibility with previous versions of PowerNet and to save RAM, global variables are used.

40.3 Parsing multipart boundaries

There are two conditions we have to deal with, parsing text lines and detecting the marker at the end of a binary file. The second situation is coded within the binary file handler.

```
: Not--          \ caddr -- flag ; true for not '--'
```

Return true if the two characters at *caddr* are not '–'.

```
: Bdry$?         \ caddr len -- 0|-1
```

Returns true if the string matches the boundary string plus two leading dashes.

```
: Boundary?      \ caddr len -- -1|0|1
```

Returns -1 if the string is a normal boundary marker, 0 if it is not a boundary marker, and 1 if it is the last boundary marker.

```
: NextBoundary   \ -- flag
```

Read text up to the next boundary marker. *Flag* is returned true if the boundary was the last one.

40.4 Flash update application

40.4.1 System interface

The flash interface with the underlying system is handled by three DEFERred words.

```
#1024 equ /FlashBlock    \ -- len
```

Unit size passed to the application.

```
Defer InitUpd    \ caddr len --
```

Initialise the app's binary update system. The input string is the file name to be received. This is passed for validation purposes and in case the file is saved in a local file system.

```
Defer AddUpd     \ caddr len --
```

Add the given memory block to the update. All blocks except the last contain `/FlashBlock` bytes of data.

`Defer TermUpd  \ --`
Terminate the app's binary update system.

40.4.2 Receiving a file

`#256 buffer: Filename$  \ -- addr`
Buffer holding file name as a counted string.

`#80 equ /BndryBlock  \ -- len`
Size of additional buffer for boundary detection. The maximum size of a boundary string is 70 characters (RFC2046). These are prefixed by CR/LF/-/- at the start and will be followed by -/- in the last boundary line.

`/FlashBlock buffer: BinBuff  \ -- addr`
Buffer for storing binary data.

`/BndryBlock buffer: BndryBuff$  \ -- addr`
Buffer for storing potential boundary string data. Holds a counted string starting with a CR.

`variable NextByte  \ -- addr`
Holds the address (in `BinBuff`) of the next character to be received.

`variable #FileSize  \ -- addr`
Holds the number of bytes stored as binary data.

`variable #AppSize  \ -- addr`
Holds the number of bytes transferred to the application.

`variable BinDone  \ -- addr`
Set if the file has been transferred.

`variable BinStatus  \ -- addr`
Holds the file receive status (0=good).

`: ResetBlock  \ --`
Reset the next byte pointer.

`: StartRxApp  \ --`
Initialise variables and buffers for file reception.

`: Block>App  \ --`
Send the data block to the application. All blocks except the last contain `/FlashBlock` bytes of data.

`: StopRxApp  \ --`
Clean up after receiving a file.

`: addData  \ byte --`
Add the byte to the binary buffer.

`: NotBoundary  \ --`
Add the partial boundary string to the data and clear the buffer.

`create CrLf--  \ -- caddr`
Holds the four characters CR/LF/-/-.

`: FullBndry?  \ -- flag`

Check the contents of `BndryBuff$` against the start of `Boundary$` which holds the received boundary separator. If the data does not match copy the data to the binary buffer. If the data matches the complete boundary string, return true, otherwise return false.
By definition the marker string does not appear in the the binary. The code reads the binary into a buffer that is 80 characters longer than the required Flash block size to accommodate the boundary string (70 chars maximum). We check for the sequence CR/LF/-/-, and if found check for the following boundary string.

`: ReceiveBinary \ --`
The separator line between the part headers and the binary file has been read. Read the binary up to the concluding part boundary. Later code will check if this is the final boundary. If the boundary tail matches the last boundary condition, return true, otherwise return false.

`: ReadBinFile   \ -- flag`
The separator line between the part headers and the binary file has been read. Read the binary up to the concluding part boundary. If the boundary string is the last one, return true, otherwise return false.

40.4.3 Part scanning

In this example, the only form part we need to process is the one containing a binary file. We detect this by parsing the headers for the pair:

```
Content-Disposition: form-data; name="appfile"; filename="foo"
Content-Type: application/octet-stream
```

We assume that a valid file is present when

- the "form-data" marker is present,
- the filename is non-null
- and the "application/octet-stream" marker is present.

`variable OctetStream    \ -- addr`
Set true when `Content-Type: application/octet-stream` has been received.

`variable FormData       \ -- addr`
Set true when the "form-data" marker has been received.

`: ?formData     \ caddr len --`
Check for the "form-data" marker.

`: ?appfile      \ caddr len --`
Check for 'name="appfile"'.

`: doPdisp       \ caddr len --`
Check the given content disposition line.

`: doPtype       \ caddr len --`
Check the given content type line.

`create PartHdrs \ -- addr`
Table holding the headers to be processed by part header handlers.

`: CheckPartHdrs \ -- x`
Check the data received in the part headers, and return x, indicating what to do with the part. If x is zero, just skip the part.

`: ProcessPart  \ -- flag`
After receiving a boundary, process a part up to and including the next boundary, returning true if it is the last one.

`: ProcessParts  \ --`
Read the parts up to the last boundary

40.5 File update handler

`: doNewApp  \ caddr len --`
This is the handler for the update request.

`' doNewApp xtPage: /NewApp.asp`
Defines the page which performs the upload.

`: .UploadDone  \ --`
Used in the body of an ASP page to indicate the response to the upload.

`MemPage: /Reflash.htm %IPSTACK%\TestPages\Reflash.htm`
A form used to request a new application.

`MemPage: /nareply.asp %IPSTACK%\TestPages\nareply.asp`
The response delivered after an upload.

40.5.1 Example for pages stored in files

This example can be used to upload binary files, e.g. a new application image, to the file system. In order to compile this code the FAT file system must be present, and the equate `FilePages?` (used in *Pages.fth*} must be set non-zero.

`Defer InitUpd  \ caddr len --`
Initialise the app's binary update system. The input string is the file name to be received. This is passed for validation purposes and in case the file is saved in a local file system.

`Defer AddUpd  \ caddr len --`
Add the given memory block to the update. All blocks except the last contain `/FlashBlock` bytes of data.

`Defer TermUpd  \ --`
Terminate the app's binary update system.

`create FileDir$ \ -- addr`
The base directory for files. This will be prepended to the page name given to `InitUpd`. The string is held as a counted string in the buffer. Make sure that your application sets `FileDir$` to point to its own pages. The directory name must not end in a separator. The directory must exist before use.

`-1 value hPostFile  \ -- handle`
Handle of the file used to save the download data.

`: InitFileUpd  \ caddr len --`
Initialise the file reception.

`: AddFileUpd  \ caddr len --`
Add the given memory block to the update. All blocks except the last contain `/FlashBlock` bytes of data.

`: TermFileUpd  \ --`
Terminate the app's binary file update system.

41 Internet RFCs

41.1 What is an RFC?

Internet standards are generally proposed and honed by various commitees in public documents across the World Wide Web. These documents are called RFCs (Request for Comments). For any standard protocol you will find a number of these documents on the Web. Some relevant ones are in the *M* directory.

41.2 Where are the latest versions?

The web site `http://www.rfc-editor.org` is currently the best source for finding RFCs.

Another starting place for a search engine; you can find them simply by entering the phrase:-

```
<protocol> +RFC
```

into the search engine. When looking up a protocol be aware of two things:-

1. Sometimes an RFC describing a protocol is made obsolescent by a later document.
2. Most protocols have extension documents posted after their initial release.

41.3 Recommended reading

TCP/IP Illustrated volume 1

W Richard Stevens

ADDISON WESLEY

TCP/IP Illustrated volume 2

Gary R Wight & W Richard Stevens

ADDISON WESLEY

InterNetworking with TCP/IP vol 2

Douglas E Comer & David L Stevens

PRENTICE HALL

Net BSD Sources

Net BSD Man Pages

The Linux sources will do fine

```
http://www.linux.org
```

42 Licence terms

42.1 Distribution of application programs

Providing that the end user has no access to the source code, compiled applications containing the PowerNet IP Code may be distributed without royalty. An acknowledgement will be gratefully appreciated. No part of the target source code may be further distributed without permission from MicroProcessor Engineering.

If you need access to an open Forth interpreter for engineering and maintenance access only, this is permitted after a (free of charge) letter of permission from MPE to ensure maintenance of our copyright. Use of the Forth interpreter for server-side scripting is permitted with the purchase of the PowerNet code.

If you need to ship open source, or wish to check what constitutes engineering and maintenance access, please contact MPE.

42.2 Warranties and support

We try to make our products as reliable and bug free as we possibly can. We support our products. If you find a bug in this product and its associated programs we will do our best to fix it. Please check first by fax or email to see if the problem has already been fixed. Please send us enough information including source code on disc or by email to us, so that we can replicate the problem and then fix it. Please also let us know the serial number of your system and its version number. We will then send you an update when we have fixed the problem. The level of technical support that we can offer may depend on the Support Policy bought with the product. Technical support will only be available on the current version of the product.

Make as many copies as you need for backup and security. The issue softwareis not copy protected. The code is copyrighted material and only ONE copy of it should be use at any one time. Contact MPE or your vendor for details of multiple copy terms and site licensing.

As this copy is sold both direct and through dealers and purchasing departments, we cannot keep track of all our users. If you send us your contact details, we will put you on our mailing list. This way we will be able to keep you informed of updates and new extensions, as they become available. If you need technical support from us we will need these details in order to respond to you. You will find the serial number of the system on the original issue discs.

Index

/

>

?

@

[

\

1

4

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z