

# QoE Ethernet Device Driver for the Q68

## User Manual

This is an Ethernet driver for the Q68 computer. It implements an IP device driver for your Local Area Network (LAN). It can be used to communicate with other Q68 computers, or QL emulators that have the IP device driver built in, and support UDP and TCP connections. Such as QPC2.

The driver sends and receives standard IP data packets, So it can also talk to other devices and systems. This implementation is not a complete IP stack, so you may find some things are not fully supported.

This driver is intended for the Q68 computer only. It will **not** work on any other QL compatible computer.

It supports UDP data packets and a pseudo TCP protocol. The TCP does not support all the usual TCP features. **It does not support errors, or lost packets.** So all the data packets must arrive, and in the correct order. This should be OK over a LAN, but you may have problems if you try to use the TCP over the internet.

The goal of this TCP implementation, is to support the IP Network driver. So the Q68 can operate a QL like network over a LAN.

DHCP can also be used to obtain an IP address automatically from a DHCP server.

Domain names are now supported. They will be resolved into IP addresses if a DNS server is available. Domain names once fetched are added to a DNS cache. This cache may be flushed, and domain names may also be added manually.

The driver will also handle ARP packets to handle MAC address requests, and Ping packets. Anything else sent to the driver is liable to be thrown away.

Many of the system calls from the QDOS IP device driver by Richard Zidlicky have been implemented in this driver. Although some features of the system calls may be missing. See the section on system calls.

QPAC2 'channels' will display information on open Ethernet channels (courtesy of routines by Wolfgang Lenerz).

The IP\_XINF system trap has been extended to accommodate DHCP and DNS.

An ETH\_IPCONFIG command has been added which lists information about the Ethernet controller and driver status.

### Acknowledgements

Thanks to Peter Graf, Martyn Hill, and Wolfgang Lenerz for their help and advice in development and testing of the driver.

## Introduction

This device driver creates an IP stack with limited functionality. So don't expect full compatibility with a real IP network.

Note that unlike TCP where the receipt of data packets is acknowledged back to the sender. The UDP protocol is very much a 'fire and forget' system, with no guarantee of the data packets getting through to the destination. The CP2200 Ethernet controller IC used in the Q68 can only store up to 8 received data packets, or 4K bytes of data (whichever comes first) before it starts to ignore received data packets.

In the background, the driver tries to keep the receive buffer in the CP2200 empty. However if data packets arrive too quickly, it may not be able to keep up. In which case data packets may get lost. If you wish to avoid this, you will need to implement your own acknowledgement system.

## Getting Started

Load the Q68NET\_BIN file. This should produce a message in #0, of

```
QoE Ethernet Driver for Q68    Vx.xx
Initializing CP2200
```

If you are running SMSQ/E, the initialization messages should be in the system language. English, French, German, Italian and Spanish are currently supported.

If you also receive a message of Self Initialization timed out or Auto-negotiation failed. Then there was a problem during initialization of the CP2200 Ethernet controller IC. If Auto-negotiation fails, then the driver will attempt to re-initialize in half duplex mode. So if you see an Auto-negotiation failed error, then the duplex mode has been set to half duplex. You can try to re-initialize the CP2200 with the **ETH\_INIT** command.

You may also use the **ETH\_ERRNO** function for a clue as to what went wrong

You now need to assign an IP Address to the Q68 with the **ETH\_SETIP** command

e.g. **ETH\_SETIP "172.16.0.20"**

Using **ETH\_SETIP** without a parameter will start a DHCP client job, That will attempt to obtain an IP address from a DHCP Server on the network.

You may want to set a subnet mask with the **ETH\_SUBNET** command, The default is 255.255.255.0

The subnet may also be set for you by the DHCP Client.

If you have a router, you can use the **ETH\_GATEWAY** command to set it. The gateway may also be set for you by the DHCP Client.

## Example usages

In these examples I am using a Q68 and a Windows computer(XP) running QPC2 V4.05. Alter the values given to suit your requirements.

### UDP

#### Q68

**LRESPR** the driver, and when initialization is complete.

**ETH\_SETIP "172.16.0.20"** This is the IP Address for the Q68.

**OPEN\_NEW#4,"udp\_172.16.0.20:5800"** Sets the Q68 to open it's network port 5800.

QPC2 (Assuming that the Windows computer is on the same network "172.16.0.xx")

**OPEN\_IN#4,"udp\_172.16.0.20:5800"** Sets QPC2 to talk to the Q68's port 5800.

On the Q68, Enter **REPeat loop:INPUT#4,a\$:PRINT LEN(a\$)!!!a\$**

Do this before the next the next step. The reason being that, it sets the Q68 waiting for input. If you tried to **PRINT#4** from the Q68 now. The Q68 will not know where to send to, As it does not know the IP address of the other computer.

On QPC2, Enter **PRINT#4,"Hello There"** This should appear on the Q68.

You may need to do this twice. For some reason QPC2 may not send the first string, It sends the Carriage Return on the end, but not the actual string.

Enter **PRINT#4,FILL\$("X",2000)** Which should also appear on the Q68.

This is sent as a fragmented packet, as it is longer than the 1472 bytes that will fit in one data packet.

Break into the loop on the Q68, and reverse the process.

In QPC2, Enter **REPeat loop:INPUT#4,a\$:PRINT LEN(a\$)!!!a\$**

and in the Q68 enter and run the program

```
100 a$="hello"  
110 FOR n = 1 TO LEN(a$)  
120 PRINT#4,a$(n);  
130 END FOR n  
140 PRINT#4
```

and then try, **a\$=FILL\$("Z",2000) : GO TO 110**

Why not **PRINT#4,"hello"**? Well **INPUT#** in the QPC2, UDP IP device driver does not seem to like more than one character at a time being sent in a data packet.

## IP Network Driver

The IP Network driver is an adaption of the standard QL Network driver, that uses a TCP connection. It can be downloaded as 'NETdriver114.zip' from ['www.dilwyn.me.uk/internet/index.html'](http://www.dilwyn.me.uk/internet/index.html)

Q68

**LRESPR** the QoE driver, and when initialization is complete.

**ETH\_SETIP "172.16.0.2"** This is the IP Address for the Q68.

**LRESPR** the IP Network driver.

**NET\_START "172.16.0.2"** This starts the IP Network driver.  
You could also use **NET\_START ETH\_GETIP\$**

This sets the Q68 as the QL Network station 2

QPC2 (Assuming that the Windows computer's IP address is "172.16.0.1")

**LRESPR** the IP Network driver.

**NET\_START "172.16.0.1"** This starts the IP Network driver running in QPC2.

This sets QPC2 as the QL Network station 1

### **FSERVE**

You now have most of the facilities of a normal QL Network. Including the **NETI** and **NETO** devices.

Q68

**DIR n1\_win1\_** Will give a directory of WIN1\_ in QPC2.

You can also use **FSERVE** on the Q68. Which would give QPC2 access to the Q68's devices.

If the last octet of the IP address of the **FSERVE** server is greater than 8, then you will need to use the **MAP\_N** command on the remote station for **FSERVE** accesses. See the IP Network driver manual for details.

## Opening Channels

I have tried as far as possible to follow the IP Device Driver as used in some QL emulators for the opening of channels

There are 3 devices available SCK\_, UDP\_, and TCP\_.

**OPEN** just creates a socket of the requested type/protocol. An IP address & port are not required.

**OPEN\_IN** creates a socket of the requested type/protocol. It sets the peer address for UDP sockets, or opens a connection to a server for TCP. The IP address and Port must be specified.

**OPEN\_NEW** creates a socket of the requested type/protocol. It sets the host address for UDP sockets. The IP Address used in the **OPEN\_NEW** command, should be the IP Address of the computer it's used on, or "0.0.0.0"

Note- It's a bit counter intuitive, but OPEN\_IN creates an output channel, and OPEN\_NEW creates an input channel. However once the connection has been established, the channels are then bi-directional. This is how the OPEN commands are defined in the QDOS IP device driver.

syntax:    *channel\_number* := *numeric\_expression*  
          *socket\_type* := **SCK\_** | **UDP\_** | **TCP\_**  
          *IP\_address* := IP Address in IPv4 numbers-and-dots notation, or a Domain name  
          *port* := Integer between 0 and 65535, or a Service name  
          *IP\_specifier* := *socket\_type\_IP\_address:port*

**OPEN**#*channel\_number,socket\_type*  
**OPEN\_IN**#*channel\_number,IP\_specifier*  
**OPEN\_NEW**#*channel\_number,IP\_specifier*

**FOPEN**([#*channel\_number*,]*socket\_type*)  
**FOP\_IN**([#*channel\_number*,]*IP\_specifier*)  
**FOP\_NEW**([#*channel\_number*,]*IP\_specifier*)

example: i. **OPEN#4,SCK\_**  
          ii. **OPEN\_IN#5,"TCP\_192.168.0.10:5800"**  
          iii. **OPEN\_NEW#ch,"UDP\_192.168.0.5:5800"**  
          vi. **OPEN\_IN#5,"TCP\_google.com:ftp"**  
          v. **OPEN\_IN#5,"TCP\_gmail.com:smtp"**  
          vi. **OPEN\_IN#5,"UDP\_129.69.1.59:https"**  
          vii. **OPEN\_IN#5,"TCP\_test1.workgroup.com:50000"**

## ETH\_INIT

## FETH\_INIT

**ETH\_INIT** will attempt to (Re)Initialize the CP2200 Ethernet controller.

The optional parameter is the required duplex mode to be used by the Ethernet controller.  
0 (default) = Auto-negotiation, 1 = Full duplex, 2 = Half duplex.

Auto-negotiation will attempt to communicate with the Ethernet controller on the other end of the connection, and both ends will agree on a speed and duplex mode to use.

**FETH\_INIT** is a function version of **ETH\_INIT** and will return 0, or an error code

Returns a Not Found error if the Ethernet driver cannot be found. And a Transmission error if Auto-negotiation fails

syntax:     *type := numeric\_expression*

**ETH\_INIT** [*type*]  
**FETH\_INIT** [(*type*)]

example:    i. **ETH\_INIT**                    Try to auto-negotiate a connection  
              i. **ETH\_INIT 1**               Set the Ethernet controller to use Full duplex  
              iii. **PRINT ETH\_INIT (2)**      Set the Ethernet controller to use Half duplex

Note: If **ETH\_INIT 0** fails, then the Ethernet controllers duplex mode is undefined. You must retry the command, or manually set the duplex mode with **ETH\_INIT 1**, or **ETH\_INIT 2**.

## ETH\_MAC\$

The function **ETH\_MAC\$** will return the MAC address of the CP2200 Ethernet controller as a dash separated string.

syntax:     **ETH\_MAC\$**

example:    **PRINT ETH\_MAC\$**

## ETH\_SETIP

**ETH\_SETIP** will set the IP Address that the Q68 will use.

Setting, or resetting the IP address will cause a gratuitous ARP request to be broadcast over the network, to inform any other computers of it's change.

Using **ETH\_SETIP** without a parameter will initiate an attempt to contact a DHCP server. If it finds that the DHCP client job is already running, It asks for confirmation to continue. Before attempting to release the currently assigned IP address and shutting down the existing DHCP client.

**ETH\_SETIP** without a parameter will wait up to 2 minutes for an allocation of an IP address from a DHCP server. This action can be aborted by pressing CTRL-SPACE.

If **ETH\_GATEWAY** or **ETH\_DNS** has not been set, then depending on the DHCP server contacted, the gateway and DNS addresses may also be set by this command.

syntax:     *IPaddress := string\_expression*

**ETH\_SETIP** [*IPaddress*]

example:    i. **ETH\_SETIP "192.168.0.10"**  
              ii. **ETH\_SETIP address\$**  
              iii. **ETH\_SETIP**

## ETH\_GETIP\$

The function **ETH\_GETIP\$** will get the IP Address that was set with the **ETH\_SETIP** command.

syntax:    **ETH\_GETIP\$**

example: i. **PRINT ETH\_GETIP\$**  
          ii. **a\$ = ETH\_GETIP\$**

## ETH\_SUBNET

**ETH\_SUBNET** will set the subnet mask that the Q68 will use.

syntax:    *mask := string\_expression*

**ETH\_SUBNET** *mask*

example: i. **ETH\_SUBNET "255.255.255.0"**  
          ii. **ETH\_SUBNET a\$**

## ETH\_SUBNET\$

The function **ETH\_SUBNET\$** will get the subnet mask that was set with the **ETH\_SUBNET** command.

syntax:    **ETH\_SUBNET\$**

example: i. **PRINT ETH\_SUBNET\$**  
          ii. **a\$ = ETH\_SUBNET\$**

## ETH\_GATEWAY

**ETH\_GATEWAY** will set the gateway (router) IP address that the Q68 will use.

syntax:    *IPaddress := string\_expression*

**ETH\_GATEWAY** *IPaddress*

example: i. **ETH\_GATEWAY "192.168.0.1"**  
          ii. **ETH\_GATEWAY a\$**

## ETH\_GATEWAY\$

The function **ETH\_GATEWAY\$** will get the gateway (router) IP address that was set with the **ETH\_GATEWAY** command, or the **ETH\_SETIP** command.

syntax:    **ETH\_GATEWAY\$**

example: i. **PRINT ETH\_GATEWAY\$**  
          ii. **a\$ = ETH\_GATEWAY\$**

## ETH\_DNS

**ETH\_DNS** will set the IP address of the DNS server that the **OPEN** command will use convert a domain name into an IP address.

syntax: *IPaddress := string\_expression*

**ETH\_DNS** *IPaddress*

example: i. **ETH\_DNS "192.168.0.1"**  
ii. **ETH\_DNS a\$**

## ETH\_DNS\$

The function **ETH\_DNS\$** will get the Domain Name Server IP address that was set with the **ETH\_DNS** command, or the **ETH\_SETIP** command.

syntax: **ETH\_DNS\$**

example: i. **PRINT ETH\_DNS\$**  
ii. **a\$ = ETH\_DNS\$**

## ETH\_NETNAME

**ETH\_NETNAME** will set the Q68's network name. The network name can be up to 26 characters long.

syntax: *netname := string\_expression*

**ETH\_NETNAME** *netname*

example: i. **ETH\_NETNAME "Ethernet Q68"**  
ii. **ETH\_NETNAME a\$**

## ETH\_NETNAME\$

The function **ETH\_NETNAME\$** will return the network name given by the **ETH\_NETNAME** command.

syntax: **ETH\_NETNAME\$**

example: i. **PRINT ETH\_NETNAME\$**  
ii. **a\$ = ETH\_NETNAME\$**

## ETH\_PING

**ETH\_PING** will send up to 4 Ping requests over the network, to the specified IP Address. The results will be sent to either the specified, or the default channel (#1).

syntax: *channel\_number := numeric\_expression*  
*IPaddress := string\_expression*

**ETH\_PING** [*#channel\_number*,] *IPaddress*

example: i. **ETH\_PING "192.168.0.1"**  
ii. **ETH\_PING#4, a\$**



## ETH\_IPCONFIG

**ETH\_IPCONFIG** will list information about the Ethernet controller and driver status to the supplied channel, or will default to channel #1.

syntax: *channel := numeric\_expression*

**ETH\_IPCONFIG** [#channel]

example: i. **ETH\_IPCONFIG**  
ii. **ETH\_IPCONFIG #2**  
iii. **ETH\_IPCONFIG #chan**

## ETH\_ERRNO

The function **ETH\_ERRNO** will return the last driver error that occurred. This is a specific error number for the Ethernet driver, and is not the same as the QDOS error that may have been displayed.

If you encounter a QDOS error message while using the driver. It may not be a very helpful error message, as there are only a limited number of QDOS error messages. Using **ETH\_ERRNO** may supply you with a more helpful error report. See the list of extended error messages at the rear of this document.

**ETH\_ERRNO** will return 0, for no current error.

After using **ETH\_ERRNO**, the last error will be cleared.

syntax: **ETH\_ERRNO**

example: i. **PRINT ETH\_ERRNO**  
ii. **a = ETH\_ERRNO**

## ARP\_ADD

**ARP\_ADD** will add an IP Address and MAC Address into the ARP table.

syntax: *IPaddress := string\_expression*  
*MACaddress := string\_expression*

**ARP\_ADD** IPaddress,MACaddress

example: i. **ARP\_ADD "192.168.0.20","01-02-03-04-05-06"**  
ii. **ARP\_ADD address\$,mac\$**

## ARP\_REMOVE

**ARP\_REMOVE** will remove an ARP table entry. If no parameter is supplied, then all the ARP table entries will be removed.

syntax: *IPaddress := string\_expression*

**ARP\_REMOVE** IPaddress

example: i. **ARP\_REMOVE "192.168.0.20"**  
ii. **ARP\_REMOVE**

## ARP\_LIST

**ARP\_LIST** will list all the ARP table entries to the supplied channel, or will default to channel #1. The list will be in the format, IP Address, MAC Address.

syntax: *channel := numeric\_expression*

**ARP\_LIST** [#*channel*]

example: i. **ARP\_LIST**  
ii. **ARP\_LIST #2**  
iii. **ARP\_LIST #chan**

## DNS\_ADD

**DNS\_ADD** will add a domain name to the DNS cache. The optional, time to live parameter defines how long in seconds that the entry in the DNS cache remains valid. The default value of 0, meaning permanent.

syntax: *domainName := string\_expression*  
*IPaddress := string\_expression*  
*ttr := numeric\_expression*

**DNS\_ADD** *domainName,IPaddress[,ttr]*

example: i. **DNS\_ADD "www.yahoo.com","87.248.100.216",30\*60**  
ii. **DNS\_ADD name\$,address\$**

## DNS\_LIST

**DNS\_LIST** will list all the entries in the DNS cache to the supplied channel, or will default to channel #1.

The list will be in the format, Domain name, IP Address, Expiry date. If there is no expiry date, then 'Never' is used in place of a date.

Note that some expiry dates may be displayed that have expired. The checking for, and removal of expired entries is only done when the cache is searched to obtain an IP address.

syntax: *channel := numeric\_expression*

**DNS\_LIST** [#*channel*]

example: i. **DNS\_LIST**  
ii. **DNS\_LIST #2**  
iii. **DNS\_LIST #chan**

## DNS\_FLUSH

**DNS\_FLUSH** will remove all the entries from the DNS cache. Except for the local host entry.

syntax: **DNS\_FLUSH**

example: **DNS\_FLUSH**

## QPAC2 Channels

If you have an Ethernet channel open. When you look at QPAC2's Channels. Information will be displayed about the open channel.

The channel information will be displayed in the format, Socket type, Local port number, Peer IP address, Peer port number

e.g. **UDP\_53760:172.16.0.9:5800**

A UDP connection, of my port 53760, to port 5800 on IP address 172.16.0.9

## Supported System Trap Calls

See the UQLX documentation for details of these system traps.

### Trap #2

D0	Name	Notes
-----		
\$01	IO_OPEN	D3=0-2
\$01	IP_ACCEPT	D3=LISTENing channel ID
\$02	IO_CLOSE	

### Trap #3

D0	Name	Notes
-----		
\$00	IO_PEND	
\$01	IO_FBYTE	
\$02	IO_FLINE	
\$03	IO_FSTRG	
\$05	IO_SBYTE	
\$07	IO_SSTRG	D2 is word sized, So should limit data size to 32K
\$48	FS_LOAD	
\$49	FS_SAVE	
\$50	IP_LISTEN	
\$51	IP_SEND	data size limited to 64K
\$52	IP_SENDTO	data size limited to 64K
\$53	IP_RECV	
\$54	IP_RECVFM	
\$58	IP_BIND	
\$59	IP_CONNECT	
\$5B	IP_GETHOSTNAME	
\$5C	IP_GETSOCKNAME	
\$5D	IP_GETPEERNAME	
\$5E	IP_GETHOSTBYNAME	
\$5F	IP_GETHOSTBYADDR	
\$64	IP_GETSERVBYNAME	
\$65	IP_GETSERVBYPORT	
\$6E	IP_GETPROTOBYNAME	
\$6F	IP_GETPROTOBYNUMBER	

\$72 IP\_INET\_ATON  
\$73 IP\_INET\_ADDR  
\$74 IP\_INET\_NETWORK  
\$75 IP\_INET\_NTOA  
\$76 IP\_INET\_MAKEADDR  
\$77 IP\_INET\_LNAOF  
\$78 IP\_INET\_NETOF

\$7C IP\_ERRNO

\$7D IP\_XINF

Notes:

IP_SEND and IP_SENDTO	D1 flag is not supported
IP_RECV and IP_RECVFM	D1 flag only supports MSG_PEEK

## New system calls

Version 0.30 of the driver introduced a new IP system trap. Note that at the moment, the operation of this system call is not set, and is liable to change.

IP\_XINF      TRAP#3      D0=7D

### Call parameters

D1  
D2.W    length of buffer  
D3.W    timeout

A0      channel ID  
A1      base of buffer  
A2  
A3

### Return parameters

D1.W    length of block returned  
D2.W    preserved  
D3.L    preserved

A0      preserved  
A1      preserved  
A2      preserved  
A3      preserved

### Error returns

NC      not complete  
NO      not open  
BO      buffer overflow

This system call will return a data block containing information about the settings of the driver.

Note, It requires an open channel. This channel can just be a SCK\_, it does not need to be connected to anything.

On systems other than this driver, some of these entries may be unavailable. It is the responsibility of the calling application to examine **inf\_driverID**, **inf\_driverVer**, and **inf\_complD** to determine which information in the data block is liable to be valid.

### System data

\$00	inf_ddbase	base of driver definition block
\$04	inf_driverID	driver ID as a 4 byte string e.g. 'Q68E'
\$08	inf_driverVer	driver version as a 4 byte string e.g. '0.29'
\$0C	inf_complD	computer ID as a 4 byte string e.g. 'Q68 '
\$10	inf_mac	6 byte system MAC address
\$16	inf_IPadd	system IP address
\$1A	inf_subnet	system subnet mask
\$1E	inf_gateway	system gateway/router address
\$22	inf_dhcp	system DHCP server IP address
\$26	inf_dns	system DNS IP address
\$2A	inf_compName	word + up to 26 bytes of the system network name
\$46	inf_txpackets	number of packets sent by the CP2200
\$48	inf_txbytes	number of bytes sent by the CP2200 *
\$4E	inf_rxpackets	number of packets received by the CP2200
\$52	inf_rxbytes	number of bytes received by the CP2200 *

### Channel data

\$56	inf_peerMac	6 byte peer MAC address
\$5C	inf_peerIP	peer IP address
\$60	inf_peerPort	peer port number
\$62	inf_sysPort	system port allocated to channel
\$64	inf_sckType	socket type 1 = TCP, 0 = UDP, -1=SCK
\$65	inf_protocol	channel protocol e.g 17 = UDP
\$66	inf_access	channel access mode (D3 on open call)
\$67	inf_sockStat	socket status

\$68    inf\_end                    end of extended info block

\* The number of bytes sent, or received by the CP2200 includes the header information.

## Ethernet driver error messages

If the Ethernet driver encounters a problem, it may store an error message that is more useful than any QDOS error message that may also be returned.

These error messages may be examined with the **ETH\_ERRNO** function.

There is only ever one error number that is stored, and it is the last error encountered.

After using the **ETH\_ERRNO** function, the error code is reset to zero.

### Receive errors

- 1 Insufficient memory to read packet into
- 2 Read packet validation failed, checksum mismatch
- 3 Received packet from unexpected MAC address
- 4 Unable to process TCP control bits
- 5 Unexpected TCP segment numbers
- 6 Unexpected TCP sequence number
  
- 8 Last packet was not transmitted successfully

### Send errors

- 10 Timeout waiting for transmit buffer to empty
- 11 Protocol not found when creating a packet
- 12 Failure sending ARP request
- 13 Failure sending Ping reply
- 14 Attempt to send too many packets with received ACK's (TCP)
- 15 Failure sending a SYN,ACK
- 16 Too many attempts made to send a packet

### Open errors

- 20 Invalid parameters for requested OPEN type
- 21 No managed ports available
- 22 Unable to acquire MAC address for specified IP address
- 23 Requested port already in use
- 24 No response from requested TCP server for a SYN request
- 25 Bad port number, or service name supplied
- 26 Bad IP address, or Domain name supplied
- 27 No colon found in supplied parameters

### I/O errors

- 30 Error creating checksum for transmission
- 31 Attempt to send more than 64K bytes
- 32 Bad sockaddr length
- 33 Fragmented packet incomplete before life ran out
- 34 I/O timeout while waiting for a MAC address
- 35 IP\_LISTEN queue out of range
- 36 Not a TCP channel
- 37 Out of memory creating a dummy channel definition block
- 38 The queue for a listening channel is full

### Close errors

- 40 Timed out waiting for a TCP close acknowledgement
- 41 While in FIN-WAIT-1, Timed out waiting for an ACK
- 42 While in FIN-WAIT-2, Timed out waiting for a FIN (last ACK)
- 43 While in LAST-ACK, Timed out waiting for final ACK
- 44 Unexpected FIN received
- 45 While in CLOSING, Timed out waiting for final ACK
- 46 Failed sending a FIN

#### IP Trap errors

50        Error reading a database entry

#### DHCP errors

60        BREAK pressed during DHCP

61        DHCP timed out

62        DHCP lease ran out

#### DNS errors

70        No DNS server setup

71        Timed out waiting for answer from DNS server

#### CP2200 Initialization errors

100       Timed out while waiting for controller to reset

110       Timed out while waiting for auto-neg in Physical layer

## Copyright and Disclaimer

This driver should not cause any problems, damage, or loss of data. However by using this device driver, you do so at your own risk, and I do not accept responsibility for any damage, or loss of data.

The driver may contain portions of, or be based on routines from the SMSQ/E source code. (Although it may be hard to find them)

### Licence for SMSQ/E

Copyright (c) 1989-2012, by

Tony Tebby  
Marcel Kilgus  
Bruno Coativy  
Fabrizio Diversi  
Phoebus Dokos  
Thierry Godefroy  
Jérôme Grimberty  
George Gwilt  
John Hall  
Mark Swift  
Per Witte  
Wolfgang Lenerz

collectively called the "COPYRIGHT HOLDERS".

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- \* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- \* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL COPYRIGHT HOLDERS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.