Using TCP/IP Sockets From BASIC

TCP = Transmission Control Protocol
IP = Internet Protocol
UDP = User Datagram Protocol

The difference between TCP and IP is that TCP is responsible for the data delivery of a packet and IP is responsible for the logical addressing. In other words, IP obtains the address and TCP guarantees delivery of data to that address.

We will not be discussing UDP here – the User Datagram Protocol (UDP) is a transport layer protocol defined for use with the IP network layer protocol. The service provided by UDP is an unguaranteed service that provides no guarantee for delivery and no protection from duplication e.g. if this arises due to software errors within an intermediate system.

The term “TCP/IP Stack” is heard a lot – it simply means a complete set of networking protocols.

The TCP/IP interface

The implementation as a device (which may be opened with the Trap #2 calls), means you can access it pretty much like other device drivers, with OPEN, CLOSE, etc.

CLOSE, INPUT#, PRINT# and other similar Trap #3 toolkit I/O commands may be used to send and fetch bytes or lines of data, using the SCK, TCP and UDP device name channels as appropriate.

The "OPEN key" is very significant - the type of OPEN (i.e. OPEN_IN, OPEN_NEW etc.) decides what can be done with the socket.

OPEN - Creates a socket of requested type or protocol (without connecting it anywhere)

OPEN_IN - TCP and UDP host and port must be specified. Open a connection TCP, or sets per address for UDP sockets

OPEN_NEW - bind TCP or UDP socket to an address. Such sockets can be used for accepting incoming connections.

For operations such as fetching HTML pages, you will normally use OPEN_IN, which is a bit counter-intuitive as you will be reading from and writing to the TCP channel.

*OPEN #3,SCK_* - opens a channel to a generic socket which can be used for accepting connections or netdb access.

OPEN #3,"TCP_host:port" - opens a TCP protocol socket, both the host and port parameter optional. host may be the page name, for example, or a numeric address such as 123.64.101.23 (completely random example) while the port number is as appropriate for the host in question.

Example:
```
OPEN_IN #3,"TCP_www.dilwyn.me.uk:80"
OPEN_IN #3,"123.64.101.23:80"
```

Opens channel #3 to the page indicated using port 80 (port 80 is normally used for HTML pages - http protocol, while port 25 would be used for SMTP email sending and port 110 is used for retrieving emails from servers, for example)

OPEN #3,"UDP_host:port" - (parameters as for TCP above)

Once the socket is opened, the actual programming depends on the nature of the service connected to, and several of the Trap #3 I/O calls may be used to send and fetch individual characters or whole lines, e.g. PRINT #, INPUT #, BGET.

You may also be able to use Martin Head's IPbasic package, available from my website at www.dilwyn.me.uk/docs/uqlx_tcp/index.html

Be aware that you may encounter the usual differences between QL and other systems such as end of line characters. It is best to regard end of line characters as CHR$(13) followed by CHR$(10) - CR LF.

Programming to download web pages, email etc do require some knowledge of network protocols - you would be well advised to study documents available on the web which explain how to use HTTP protocols, for example. HTTP stands for Hypertext Transfer Protocol. The examples given here use HTTP 1.0

Here's how to open a TCP channel to fetch a page called index.html from a website called www.myexample.com

First, the TCP channel is opened using OPEN_IN or FOP_IN from Toolkit 2:

```
OPEN_IN #3,"TCP_www.myexample.com:80"
```
(don't put a "http:" in front of the "www" in the OPEN_IN statement, it won't work, http is specified in the following GET command)

Then use a GET command to tell the server what you want to fetch:

```
PRINT #3,"GET http://www.myexample.com/index.html HTTP/1.0"&CHR$(13)
```

Since many modern servers are private/virtual ones, we need a HOST command to ensure the page can be found correctly (e.g. where two separate websites reside on the same server).

```
PRINT #3,"HOST:http://www.myexample.com" & CHR$(13)
```

Finally, we tell the server that we have ended sending it commands by use of a single blank line:

```
PRINT #3,CHR$(13)
```

The server will now start to send you information. This is the response header, again terminated by a blank line. It may include several lines.

The first line of a response is the status line. This consists of the protocol version, a numeric status code and matching phrase of text message, each separated by a space:

```
HTTP/1.0 200 OK
```

The first digit of the status code defines the class of response. They are not "error codes" as such, but they may indicate something went wrong with whatever was asked of the server.

1xx: Informational, reserved for future use

2xx: Success, the action was successfully received, understood and accepted.
200 - OK
201 - Created
202 - Accepted
203 - No content

3xx: Redirection - further action must be taken in order to complete the request.
301 - Moved permanently
302 - Moved temporarily
304 - Not modified

4xx: Client Error - the request contains bad syntax or cannot be fulfilled.
401: Unauthorised
403: Forbidden

404: Not Found

5xx: Server error - the server failed to fulfil an apparently valid request.
500: Internal Server Error
501: Not Implemented
502: Bad Gateway
503: Service Unavailable

Other 3 digit extension codes may be encountered. The client program should treat similarly any status codes received with the same first digit, e.g. a code starting with "4" usually indicates something wrong with the request made of the server.

Entity header fields describe some attribute of the data to be received.

One we are particularly interested in is the one which states what the "Content Length" will be, in other words, how many bytes of data we can expect to download.

Since it isn't always possible to rely on EOF to detect the end of data from a TCP channel, we need to know the explicit length of the data to receive.

Statements like this are at the start of a received line, followed by colons, so you can read them using lines like 320 to read them. Examples of entity headers which may be received when using HTTP/1.0

Allow:
Authorisation:
Content-Encoding:
Content-Length:
Content-Type:
Date:
Expires:
From:
If-Modified-Since:
Last-Modified:
Location:
Pragma:
Referer:
Server:
User-Agent:
WWW-Authenticate:

See HTTP documentation such as RFC1945 for details of these.

Here is a short example program to download a single page from my website – it ends up as a file called history_html on RAM1_. The program removes carriage returns from the file - remove "IF k$ <> CHR$(13) from line 440 if this is not required. The loop from line 260 to 350 may be used to extract and handle these, in this example it only reads the Content-Length.

```
100 REMark example program to download a page called history.html
110 REMark from my website at www.dilwyn.me.uk
120 :
130 CLS : CLS #0
140 PRINT #0,'Talking to server...'
150 OPEN_IN #3,"tcp_www.dilwyn.me.uk:80"
160 PRINT #3,"GET http://www.dilwyn.me.uk/history.html HTTP/1.0"&CHR$(13)
170 PRINT #3,"HOST:http://www.dilwyn.me.uk"&CHR$(13)
180 PRINT #3,CHR$(13) : REMark blank line to end section
190 :
200 REMark open a temporary file on ramdisk to hold the page fetched
210 OPEN_NEW #4,ram1_history_html
```

```
220 :
230 REMark read back header section
240 REMark don't send header dialogue to temporary file
250 contentLength = 0 : REMark length of body to fetch
260 REPeat loop
270   IF EOF(#3) : EXIT loop
280   INPUT #3,t$
290   IF t$ = CHR$(13) THEN EXIT loop : REMark blank line ends header
300   IF t$(LEN(t$)) = CHR$(13) THEN t$ = t$(1 TO LEN(t$)-1)
310   PRINT t$ : REMark show on screen
320   IF ("Content-Length:" INSTR t$) = 1 THEN
330     contentLength = t$(16 TO LEN(t$))
340   END IF
350 END REPeat loop
360 PRINT : REMark blank line between header and body
370 :
380 REMark read the body part (the actual HTML page)
390 REMark copy into temporary file with PRINT #4 statements
400 PRINT #0,'Receiving page...(';contentLength;' bytes)'
410 FOR a = 1 TO contentLength
420   k$ = INKEY$(#3)
430   REMark strip any carriage return codes for QL.
440   IF k$ <> CHR$(13) : PRINT k$; : PRINT #4,k$;
450 END FOR a
460 :
470 CLOSE #4
480 CLOSE #3
```

Here is an example small program written in BASIC to read email headers. The program was written by Jon Dent, author of the "soql" package, slightly modified to use OPEN_IN instead of OPEN for the emulators:

```
100 OPEN_IN #8,"tcp_mail.isp.net:110": REM your account, smtp port 110
110 inst$ = "" : stage = 0
120 crlf$ = CHR$(13) & CHR$(10)
130 REPeat
140   a$ = INKEY$(#8,100)
150   IF a$ <> "" THEN inst$ = inst$ & a$
160   PRINT a$;
170   SELect ON stage
180     =0:
190        IF "OK" INSTR inst$ THEN
200          inst$= ""
210          PRINT #8, "USER user.name.here";crlf$;
220          stage= 1
230        END IF
240     =1:
250        IF "OK" INSTR inst$ THEN
260          inst$= ""
270          PRINT #8, "PASS password.here";crlf$;
280          stage= 2
290        END IF
300     =2:
310        IF "OK" INSTR inst$ THEN
320          inst$= ""
330          PRINT #8, "LIST";crlf$;
340          stage= 3
350        END IF
360     =3:
370        IF "OK" INSTR inst$ THEN
```

```
380          PRINT #0,"view which mail number ? 0 to QUIT":
390          INPUT #0,number$
400          IF number$<>0
410            PRINT #8,"TOP ";number$;" 5";crlf$;
420          ELSE
430            PRINT #8,"QUIT";crlf$;
440          END IF
450          CLS:CLS #0
460          stage= 2
470        END IF
480   END SELect
490 END REPeat
```