QL

Concepts

The Concept Reference Guide describes concepts relating to SuperBASIC and the QL hardware. It is best to think of the Concept Guide as a source of information. If there are any questions about SuperBASIC or the QL itself which arise out of using the computer or other sections of the manual then the Concept Guide may have the answer. Concepts are listed in alphabetical order using the most likely term for that concept. If the subject cannot be found then consult the index which should be able to tell you which page to turn to.

Where an example is listed with line numbers, then it is a complete program and can be entered and run. Examples listed without numbers are usually simple commands and it may not always be sensible to enter them into the computer in isolation. Examples which demonstrate stipples will not work properly on a television set.

by Stephen Berry *(Sinclair Research Limited)*

Index

Arrays

Arrays must be **DIM**ensioned before they are used. When an array is dimensioned the value of each of its elements is set to zero or a zero length string if it is a string array. An array dimension runs from zero up to the specified value. There is no limits to the number of dimensions which can be defined other than the total memory capacity of the computer. An array of data is stored such that the last index defined cycles round most rapidly:

Example:

the array defined by

```
DIM array(2,4)
```

will be stored as

0,0 low address
0,1
0,2
0,3
0,4
1,0
1,1
1,3
1,4
2,0
2,1
2,2
2,3
2,4 high address

The element referred to by **array(a,b,c)** is equivalent to the element referred to by **array(a)(b)(c)**

Command	Function
DIM	dimension an array
DIMN	find out about the dimensions of an array

BASIC

SuperBASIC includes most of the functions, procedures and constructs found in other dialects of BASIC. Many of these functions are superfluous in SuperBASIC but are included for compatibility reasons:

GOTO	use IF, REPEAT, etc
GOSUB	use DEFine PROCedure
ON...GOTO	use SELect
ON...GOSUB	use SELect

Some commands appear not to be present. They can always be obtained by using a more general function. For example, there are no **LPRINT** or **LLIST** statements in SuperBASIC but output can be directed to a printer by opening the relevant channel and using **PRINT** or **LIST**.

LPRINT	use PRINT #
LLIST	use LIST #
VAL	not required in SuperBASIC
STR$	not required in SuperBASIC
IN	not applicable to 68008 processor
OUT	not applicable to 68008 processor

comment

Almost all forms of BASIC require the **VAL(x$)** and **STR$(x)** functions in order to be able to convert the internal codified form of the value of a string expression to or from the internal codified form of the value of a numeric expression.

These functions are redundant in SuperBASIC because of the provision of a unique facility referred to as "coercion". The **VAL** and **STR$** functions are therefore not provided.

Break

If at any time the computer fails to respond or you wish to stop a SuperBASIC program or command then

hold down

CTRL

and then press

SPACE

A program broken into in this way can be restarted by using the **CONTINUE** command.

Channels

A *channel* is a means by which data can be output to or input from a QL *device*. Before a channel can be used it must first be activated (or opened) with the **OPEN** command. Certain channels should always be kept open: these are the default channels and allow simple communication with the QL via the keyboard and screen. When a channel is no longer in use it can be deactivated (closed) with the **CLOSE** command.

A channel is identified by a channel number. A channel number is a numeric expression preceded by a #. When the channel is opened a device is linked to a channel number and the channel is initialised. Thereafter the channel is identified only by its channel number. For example:

```
OPEN #5,SER1
```

Will link serial port 1 to the channel number 5. When a channel is closed only the channel number need be specified. For example:

```
CLOSE #5
```

Opening a channel requires that the *device driver* for that channel be activated. Usually there is more than one way in which the device driver can be activated, for example the network requires a *station number*. This extra information is appended to the device name and passed to the **OPEN** command as a parameter. See concepts *device* and *peripheral expansion*.

Data can be output to a channel by **PRINT**ing to that channel; this is the same mechanism by which output appears on the QL screen. **PRINT** without a parameter outputs to the default channel #1. For example:

```
10 OPEN #5,mdv1_test_file
20 PRINT #5,"this text is in file test_file"
30 CLOSE #5
```

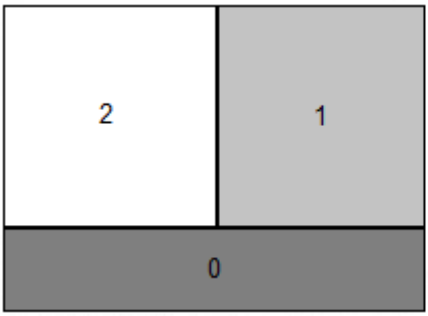
will output the text "this text is in file test_file" to the file test_file. It is important to close the file after all the accesses have been completed to ensure that all the data is written.

Data can be input from a file in an analogous way using **INPUT**. Data can be input from a channel a character at a time using **INKEY$**

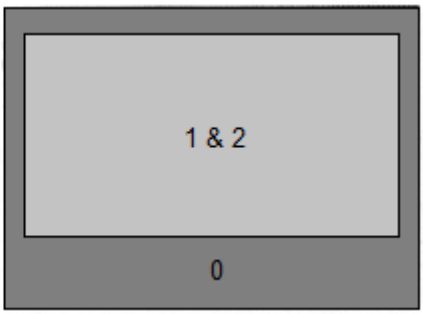
A channel can be opened as a console channel; output is directed to a specified window on the QL screen and input is taken from the QL keyboard. When a console channel is opened the size and shape of the initial window is specified. If more than one console channel is active then it is possible for more than one channel to be requesting input at the same time. In this case, the required channel can
be selected by pressing CTRL C to cycle round the waiting channels. The cursor in the window of the selected channel will flash.

The QL has three default channels which are opened automatically. Each of these channels is linked to a window on the QL screen.

channel 0 - command and error channel
channel 1 - output and graphics channel
channel 2 - program listing channel



Monitor



Television

Command	Function
OPEN	open a channel for I/O
CLOSE	close a previously opened channel
PRINT	output to a channel
INPUT	input from a channel
INKEY$	input a character from a channel

Character set and keys

The cursor controls are not built in to the operating system: however, if these functions are to be provided by applications software, they should use the keys specified; also the specified keys should not normally be used for any other purpose.

Decimal	Hex	Keying	Display/Function
0	00	CTRL £	NULL
1	01	CTRL A	
2	02	CTRL B	
3	03	CTRL C	Change input channel (see note)
4	04	CTRL D	
5	05	CTRL E	
6	06	CTRL F	
7	07	CTRL G	

8	08	CTRL H	
9	09	TAB (CTRL I)	Next field
10	0A	ENTER (CTRL J)	New line / Command entry
11	0B	CTRL K	
12	0C	CTRL L	
13	0D	CTRL M	Enter
14	0E	CTRL N	
15	0F	CTRL O	
16	10	CTRL P	
17	11	CTRL Q	
18	12	CTRL R	
19	13	CTRL S	
20	14	CTRL T	
21	15	CTRL U	
22	16	CTRL V	
23	17	CTRL W	
24	18	CTRL X	
25	19	CTRL Y	
26	1A	CTRL Z	
27	1B	ESC (CTRL SHIFT |)	Abort current level of command
28	1C	CTRL SHIFT \	
29	1D	CTRL SHIFT]	
30	1E	CTRL SHIFT ´	
31	1F	CTRL SHIFT ESC	
32	20	SPACE	
33	21	SHIFT 1	!
34	22	SHIFT '	"
35	23	SHIFT 3	#
36	24	SHIFT 4	$
37	25	SHIFT 5	%
38	26	SHIFT 7	&
39	27	'	'
40	28	SHIFT 9	(
41	29	SHIFT 0	)
42	2A	SHIFT 8	*
43	2B	SHIFT =	+
44	2C	,	,
45	2D	-	-
46	2E	.	.
47	2F	/	/
48	30	0	0
49	31	1	1
50	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5
54	36	6	6

55	37	7	7
56	38	8	8
57	39	9	9
58	3A	SHIFT ;	:
59	3B	;	;
60	3C	SHIFT ,	<
61	3D	=	=
62	3E	SHIFT .	>
63	3F	SHIFT /	?
64	40	SHIFT 2	@
65	41	SHIFT A	A
66	42	SHIFT B	B
67	43	SHIFT C	C
68	44	SHIFT D	D
69	45	SHIFT E	E
70	46	SHIFT F	F
71	47	SHIFT G	G
72	48	SHIFT H	H
73	49	SHIFT I	I
74	4A	SHIFT J	J
75	4B	SHIFT K	K
76	4C	SHIFT L	L
77	4D	SHIFT M	M
78	4E	SHIFT N	N
79	4F	SHIFT O	O
80	50	SHIFT P	P
81	51	SHIFT Q	Q
82	52	SHIFT R	R
83	53	SHIFT S	S
84	54	SHIFT T	T
85	55	SHIFT U	U
86	56	SHIFT V	V
87	57	SHIFT W	W
88	58	SHIFT X	X
89	59	SHIFT Y	Y
90	5A	SHIFT Z	Z
91	5B	[	[
92	5C	\	\
93	5D	]	]
94	5E	SHIFT 6	^
95	5F	SHIFT -	_
96	60	£	£
97	61	A	a
98	62	B	b
99	63	C	c
100	64	D	d
101	65	E	e

102	66	F	f
103	67	G	g
104	68	H	h
105	69	I	i
106	6A	J	j
107	6B	K	k
108	6C	L	l
109	6D	M	m
110	6E	N	n
111	6F	O	o
112	70	P	p
113	71	Q	q
114	72	R	r
115	73	S	s
116	74	T	t
117	75	U	u
118	76	V	v
119	77	W	w
120	78	X	x
121	79	Y	y
122	7A	Z	z
123	7B	SHIFT [	{
124	7C	SHIFT \	|
125	7D	SHIFT]	}
126	7E	SHIFT ´	~
127	7F	SHIFT ESC	©
128	80	CTRL ESC	ä
129	81	CTRL SHIFT 1	ã
130	82	CTRL SHIFT '	â
131	83	CTRL SHIFT 3	é
132	84	CTRL SHIFT 4	ö
133	85	CTRL SHIFT 5	õ
134	86	CTRL SHIFT 7	ø
135	87	CTRL '	ü
136	88	CTRL SHIFT 9	ç
137	89	CTRL SHIFT 0	ñ
138	8A	CTRL SHIFT 8	æ
139	8B	CTRL SHIFT =	œ
140	8C	CTRL ,	á
141	8D	CTRL _	à
142	8E	CTRL .	â
143	8F	CTRL /	ë
144	90	CTRL 0	è
145	91	CTRL 1	ê
146	92	CTRL 2	ï
147	93	CTRL 3	í
148	94	CTRL 4	ì

149	95	CTRL 5	î
150	96	CTRL 6	ó
151	97	CTRL 7	ò
152	98	CTRL 8	ô
153	99	CTRL 9	ú
154	9A	CTRL SHIFT ;	ù
155	9B	CTRL ;	û
156	9C	CTRL SHIFT ,	ß
157	9D	CTRL =	¢
158	9E	CTRL SHIFT .	¥
159	9F	CTRL SHIFT /	`
160	A0	CTRL SHIFT 2	Ä
161	A1	CTRL SHIFT A	Ã
162	A2	CTRL SHIFT B	Â
163	A3	CTRL SHIFT C	É
164	A4	CTRL SHIFT D	Ö
165	A5	CTRL SHIFT E	Õ
166	A6	CTRL SHIFT F	▯
167	A7	CTRL SHIFT G	Ü
168	A8	CTRL SHIFT H	Ç
169	A9	CTRL SHIFT I	▯
170	AA	CTRL SHIFT J	Æ
171	AB	CTRL SHIFT K	Œ
172	AC	CTRL SHIFT L	α
173	AD	CTRL SHIFT M	δ
174	AE	CTRL SHIFT N	θ
175	AF	CTRL SHIFT O	λ
176	B0	CTRL SHIFT P	μ
177	B1	CTRL SHIFT Q	π
178	B2	CTRL SHIFT R	Φ
179	B3	CTRL SHIFT S	¡
180	B4	CTRL SHIFT T	¿
181	B5	CTRL SHIFT U	Ƨ
182	B6	CTRL SHIFT V	§
183	B7	CTRL SHIFT W	¤
184	B8	CTRL SHIFT X	«
185	B9	CTRL SHIFT Y	»
186	BA	CTRL SHIFT Z	°
187	BB	CTRL [	÷
188	BC	CTRL \	←
189	BD	CTRL]	→
190	BE	CTRL SHIFT 6	↑
191	BF	CTRL SHIFT _	↓
192	C0	Left	Cursor left one character
193	C1	ALT Left	Cursor to start of line
194	C2	CTRL Left	Delete left one character
195	C3	CTRL ALT Left	Delete line

196	C4	SHIFT Left	Cursor left one word
197	C5	SHIFT ALT Left	Pan left
198	C6	SHIFT CTRL Left	Delete left one word
199	C7	SHIFT CTRL ALT Left	
200	C8	Right	Cursor right one character
201	C9	ALT Right	Cursor to end of line
202	CA	CTRL Right	Delete character under cursor
203	CB	CTRL ALT Right	Delete to end of line
204	CC	SHIFT Right	Cursor right one word
205	CD	SHIFT ALT Right	Pan right
206	CE	SHIFT CTRL Right	Delete word under & right of cursor
207	CF	SHIFT CTRL ALT Right	
208	D0	Up	Cursor right
209	D1	ALT Up	Scroll up
210	D2	CTRL Up	Search backward
211	D3	ALT CTRL Up	
212	D4	SHIFT Up	Top of screen
213	D5	SHIFT ALT Up	
214	D6	SHIFT CTRL Up	
215	D7	SHIFT CTRL ALT Up	
216	D8	Down	Cursor down
217	D9	ALT Down	Scroll down
218	DA	CTRL Down	Search forwards
219	DB	ALT CTRL Down	
220	DC	SHIFT Down	Bottom of screen
221	DD	SHIFT ALT Down	
222	DE	SHIFT CTRL Down	
223	DF	SHIFT CTRL ALT Down	
224	E0	CAPS LOCK	Toggle CAPS LOCK function
225	E1	ALT CAPS LOCK	
226	E2	CTRL CAPS LOCK	
227	E3	ALT CTRL CAPS LOCK	
228	E4	SHIFT CAPS LOCK	
229	E5	SHIFT ALT CAPS LOCK	
230	E6	SHIFT CTRL CAPS LOCK	
231	E7	SHIFT CTRL ALT CAPS LOCK	
232	E8	F1	
233	E9	CTRL F1	
234	EA	SHIFT F1	
235	EB	CTRL SHIFT F1	
236	EC	F2	
237	ED	CTRL F2	
238	EE	SHIFT F2	
239	EF	CTRL SHIFT F2	
240	F0	F3	
241	F1	CTRL F3	
242	F2	SHIFT F3	

243	F3	CTRL SHIFT F3	
244	F4	F4	
245	F5	CTRL F4	
246	F6	SHIFT F4	
247	F7	CTRL SHIFT F4	
248	F8	F5	
249	F9	CTRL F5	
250	FA	SHIFT F5	
251	FB	CTRL SHIFT F5	
252	FC	SHIFT space	"Special" space
253	FD	SHIFT TAB	Back tab (CTRL ignored)
254	FE	SHIFT ENTER	"Special" newline (CTRL ignored)
255	FF	See below	

Codes up to 20 hex are either control characters or non-printing characters. Alternative keyings are shown in brackets after the main keying.

Note that CTRL-C is trapped by Qdos and cannot be detected without changes to the system variables.

Note that codes C0-DF are cursor control commands.

The ALT key depressed with any key combination other than cursor keys or CAPS LOCK generates the code FF, followed by a byte indicating what the keycode would have been if ALT had not been depressed.

Note that CAPS LOCK and CTRL-F5 are trapped by Qdos and cannot be detected without special software.

Clock

The QL contains a real time clock which runs when the computer is switched on.

The format used for the date and time is standard ISO format.

```
1983 JAN 01 12:09:10
```

Individual year, month, day and time can all be obtained by assigning the string returned by **DATE** to a *string variable* and *slicing* it. The clock will run from 1961 JAN 01 00:00:00

Comment:

For a description of the format, see BS5249: Part 1: 1976 and as modified in Appendix D.2.1 Table 5 Serial 5 and Appendix E.2 Table 6 Serials 1 and 2.

Command	Function
SDATE	set the clock
ADATE	adjust the clock
DATE	return the date as a number
DATE$	return the date as a string
DAY$	return day of the week

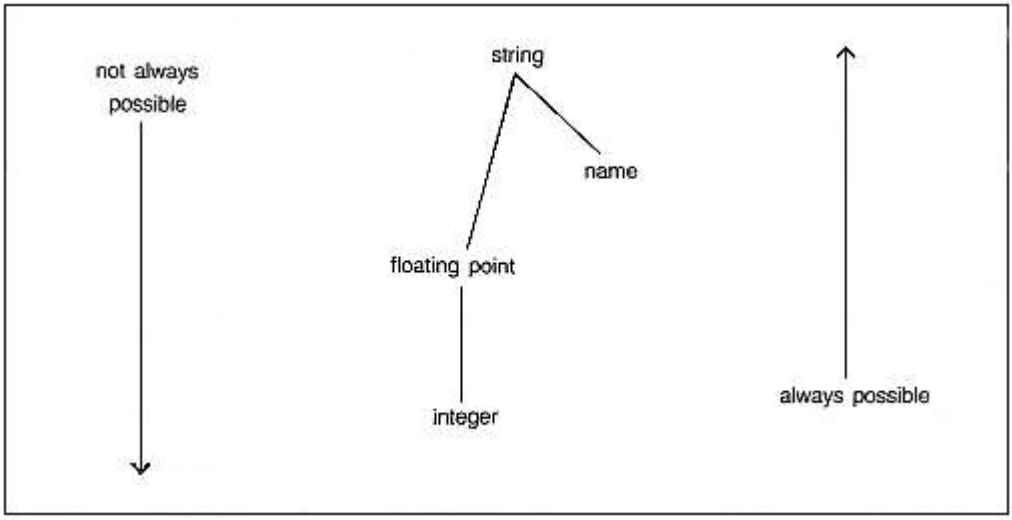
Coercion

If necessary SuperBASIC will convert the type of unsuitable data to a type which will allow the specified operation to proceed.

The *operators* used determine the conversion required. For example, if an operation requires a *string* parameter and a numeric parameter is supplied then SuperBASIC will first convert the parameter to type string. It is not always possible to convert data to the required form and if the data cannot be converted an error is reported.

The type of a *function or procedure* parameter can also be converted to the correct type. For example, the SuperBASIC **LOAD** command requires a parameter of type *name* but can accept a parameter of type *string* and which will be converted to the correct type by the procedure itself. Coercion of this form is always dependent on the way the function or procedure was implemented.

There is a natural ordering of data types on the QL, see figure below. *String* is the most general type since it can represent names, floating point and *integer* numbers. *Floating* point is not as general as string but is more general than integer since floating point data can represent integer (almost exactly). The figure below shows the ordering diagramatically. Data can always be converted moving up the diagram but it is not always possible moving down.



Example

`a = b + c`	(no conversion is necessary before performing the addition. Conversion is not necessary before assigning the result to a)
`a% = b + c`	(no conversion is necessary before performing the addition but the result is converted to integer before assigning)
`a$ = b$ + c$`	(**b$** and **c$** are converted to floating point, if possible, before being added together. The result is converted to string before assigning)
`LOAD "mdv1_data"`	(the string **"mdv1_data"** is converted to type name by the **LOAD** procedure before it is used)

Statements can be written in SuperBASIC which would generate errors in most other computer languages. In general, it is possible to mix data types in a very flexible manner:

i. `PRINT "1" + 2 + "3"`
ii. `LET a$ = 1 + 2 + a$ + "4"`

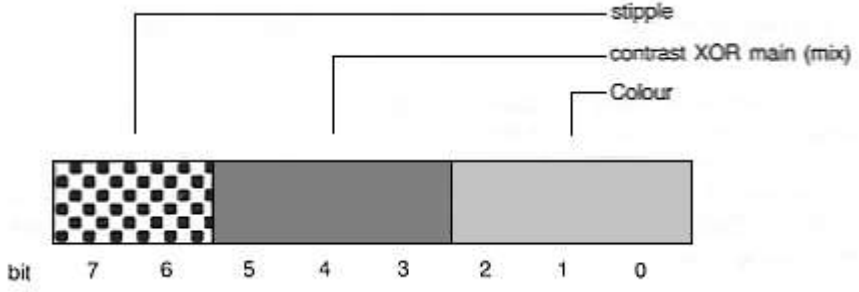
COLOUR

Colours on the QL can be either a **solid colour** or a **stipple** - a mixture of two colours to some predefined pattern. Colour specification on the QL can be up to three items: a colour, a contrast colour and a stipple pattern.

Single:

colour:= composite_colour

The single argument specifies the three parts of the colour specification. The main colour is contained in the bottom three bits of the colour byte. The next three bits contain the exclusive or (XOR) of the main colour and the contrast colour. The top two bits indicate the *stipple* pattern.



By specifying only the bottom three bits (i.e. the required colour) no *stipple* will be requested and a single solid colour will be used for display.

Double:

colour: = background, contrast

The colour is a *stipple* of the two specified colours. The default checkerboard stipple is assumed (stipple 3)

Triple:

colour: = background, contrast, stipple

Background and *contrast* colours and *stipple* are each defined separately.

Colours:

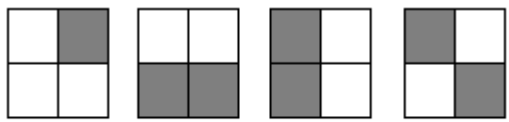
The codes for colour selection depend on the screen mode in use:

Code	bit pattern	composition			colour	
					8 colour	**4 colour**
0	0 0 0				black	black
1	0 0 1			blue	blue	black
2	0 1 0		Red		red	red
3	0 1 1		red +	blue	magenta	red
4	1 0 0	green			green	green
5	1 0 1	green +		blue	cyan	green
6	1 1 0	green +	Red		yellow	white
7	1 1 1	green +	red +	blue	white	white

Colour Composition and Codes

Stipples

Stipples mix a background and a contrast colour in a fine stipple pattern. Stipples can be used on the QL in the same manner as ordinary solid colours although stipples may not be reproduced correctly on an ordinary domestic television. There are four stipple patterns:



Stipple 0 Stipple 1 Stipple 2 Stipple 3

Stipple 3 is the default.

Example:

i. `PAPER 255 : CLS`
ii. `PAPER 2,4 : CLS`
iii. `PAPER 0,2,0 : CLS`

Warning:

Stipples may not reproduce correctly on a domestic television set which is fed via the UHF socket.

COMMUNICATIONS RS-232-C

The QL has two serial ports (called SER1 and SER2) for connecting it to equipment which uses serial communications obeying EIA standard RS-232-C or a compatible standard.

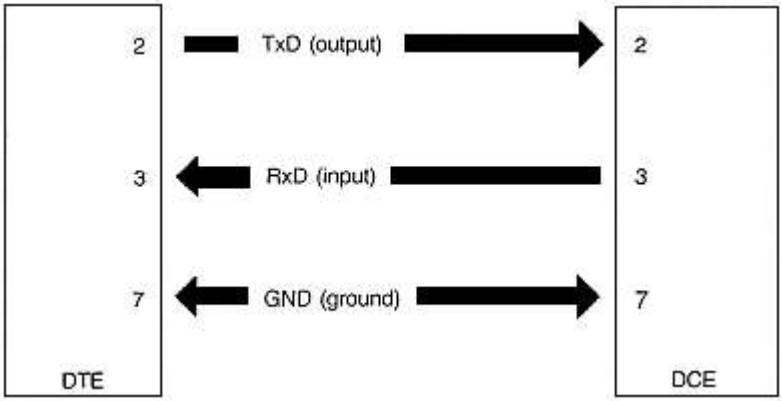
The RS-232-C 'standard' was originally designed to enable computers to send and receive data via telephone lines using a modem. However, it is now frequently used to connect computers directly with each other and to various items of peripheral equipment, e.g. printers, plotters, etc.

As the RS-232-C 'standard' manifests itself in many different forms on different pieces of equipment, it can be an extremely difficult job, even for an expert to connect together for the first time two pieces of supposedly standard RS-232-C equipment. This section will attempt to cover most of the basic problems that you may encounter.

The RS-232-C 'standard' refers to two types of equipment:

1. Data Terminal Equipment (DTE)
2. Data Communication Equipment (DCE)

The standard envisaged that the terminal (usually the DTE) and the modem (usually the DCE) would both have the same type of connector.



The diagram above illustrates how the DTE transmits data on pin 2 whilst the DCE must receive data on its pin 2 (which is still called transmit data!). Likewise, the DTE receives data on pin 3 whilst the DCE must transmit data on its pin 3 (which is still called receive data!). Although this is confusing in itself, it can lead to far greater problems when there is disagreement as to whether a certain device should be configured as DCE or DTE.

Unfortunately, some people decide that their computers should be configured as DCE devices whilst others configure equivalent computers as DTE devices. This obviously leads to difficulties in the configuration of the serial ports on each piece of equipment.

SER1 on the QL is configured as DCE, while SER2 is configurd as DTE. Therefore, it should be possible to connect at least one of the serial ports to a given device simply by using whichever port is wired the 'correct' way. The pin-out for the serial ports is given below. A cable for connecting the QL to a standard 25-way 'D' type connector is available from Sinclair Research Limited.

SER1		
pin	**name**	**function**
1	GND	Signal ground
2	TxD	Input
3	RxD	Output
4	DTR	Ready input
5	CTS	Ready output
6	-	+12V

TxD	Transmit Data
RxD	Receive Data

SER2		
pin	**name**	**function**
1	GND	Signal ground
2	TxD	Output
3	RxD	Input
4	DTR	Ready output
5	CTS	Ready input
6	-	+12V

DTR	Data Terminal Ready
CTS	Clear To Send

Once the equipment has been connected to the 'correct' port, the baud rate (the speed of transmission of data) must be set so that the baud rates for both the QL and the connected equipment are the same. The QL can be set to operate at:

75
300
600
1200
2400
4800
9600
19200 (transmit only) baud

The QL baud rate is set by the **BAUD** command and is set for both channels. The baud rates cannot be set independently.

The **parity** to be used by the QL must also be set to match that expected by the peripheral equipment. Parity is usually used to detect simple transmission errors and may be set to be even, odd, mark, space or no parity, i.e. all 8 bits of the byte are used for data.

Stop bits mark the end of transmission of a byte or character. The QL will receive data with one, one and a half, or two stop bits, and will always transmit data with at least two stop bits. Note that if the QL is set up to 9600 baud it will not receive data with only one stop bit: at least one and a half stop bits are required.

The may be necessary to connect the **handshake** lines between the QL and a piece of equipment connected to it. This allows the QL and its peripheral to monitor and control their rate of communication. They may need to do this if one of them cannot cope with the speed at which data is being transmitted. The QL uses two handshaking lines:

CTS	Clear to Send
DTR	Terminal Ready

If DTE cannot cope with the rate of transmission of data then it can negate the DTR line which tells the DCE to stop sending data. Obviously, when the DTE has caught up it tells the DCE, via the DTR line, to start transmitting again. In the same way, the DCE can stop the DTE sending data by negating the CTS line. If additional control signals are required they can be wired up using the 12V supply available on both serial ports.

Although transmission from the QL is often possible without any handshaking at all, the QL will not receive correctly under any circumstances without the use of CTS on SER1 and DTR on SER2.

Communications on the QL are 'full duplex', that is both receive and transmit can operate concurrently.

The parity and handshaking are selected when the serial channel is opened.

command	Function
BAUD	set transmission speed
OPEN	open serial channels *
CLOSE	close serial channels

* see concept 'DEVICE' for a full specification

DATA TYPES - VARIABLES

integer

Integers are whole numbers in the range -32768 to +32767. *Variables* are assumed to be integer if the variable identifier is suffixed with a percent %. There are no integer constants in SuperBASIC, so all constants are stored as *floating point numbers*.

syntax: *identifier%*

example:
i. `counter%`
ii. `size_limit%`
iii. `this_is_an_integer_variable%`

floating point

Floating point numbers are in the range +/- (10^{-615} to 10^{+615}), with 8 signiflcant digits. Floating point is the default data type in SuperBASIC. All constants are held in floating point form and can be entered
using exponent notation.

syntax: *identifier* | *constant*

example:
i. `current_accumulation`
ii. `76.2356`
iii. `354E25`

string

A string is a sequence of characters up to 32766 characters long. *Variables* are assumed to be type string if the variable name is suffixed by a $. String data is represented by enclosing the required characters in either single or double quotation marks.

syntax: *identifier$* | *"text"*

example:
i. `string_variables$`
ii. `"this is string data"`
iii. `"this is another string"`

name

Type name has the same form as a standard SuperBASIC *identifier* and is used by the name system to name *Microdrive files* etc.

syntax: *Identifier*

example: i. `mdv1_data_file`
ii. `ser1e`

DEVICES

A **device** is a piece of equipment on the QL to which data can be sent (input) and from which data can be output.

Since the system makes no assumptions about the ultimate I/O (input/output) device which will be used, the I/O device can be easily changed and the data diverted between devices. For example, a *program* may have to output to a printer at some point during its run. If the printer is not available then the output can be diverted to a *Microdrive file* and stored. The file can then be printed at a later date. I/O on the QL can be thought of as being written to and read from a **logical file** which is in a standard device-independent form.

All device specific operations are performed by individual **device drivers** specially written for each device on the QL. The system can automatically find and include drivers for peripheral devices which are fitted. These should be written in the standard QL device driver format; see the concept *peripheral expansion*.

When a device is activated a *channel* is opened and linked to the device. To correctly open a channel device basic information must sometimes be supplied. This extra information is appended to the device name.

The file name should conform to the rules for a SuperBASIC *type name* though it is also possible to build up the file name (device name) as a SuperBASIC *string expression*.

In summary the general form of a file name is:

identifier [information]

where the complete file name (including the extra information) conforms to the rules for a SuperBASIC identifier.

Each logical device on the system requires its own particular 'extra information' although default parameters will be assumed in each case where possible.

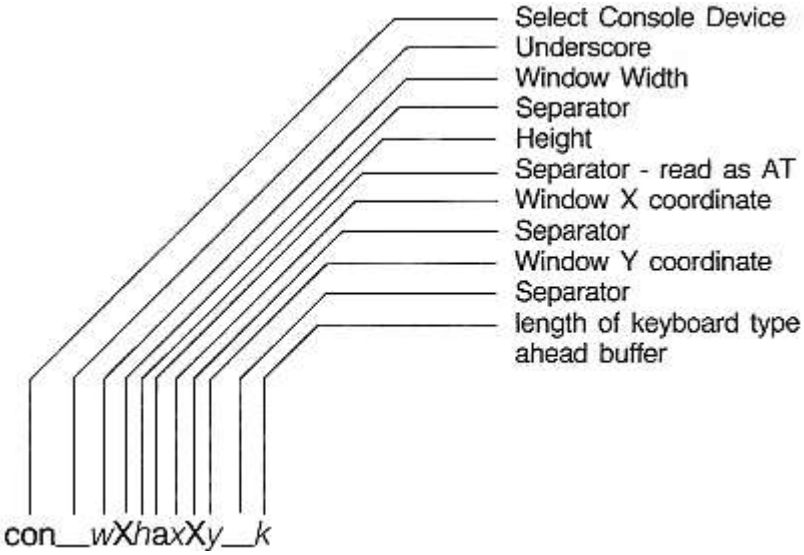
Define

device: = name

where the form of the device name is outlined below.

example

for console device



CON_*w***X***h***a***x***X***y*_*k*

Console I/O

[*w***X***h*] - window *width, height*
[**A***x***X***y*] - window *X,Y* coordinate of upper left-hand corner
[*k*] - keyboard type ahead buffer length (bytes)

default: con_448x180a32x16_128

example:
```
OPEN #4,con_20x50a0x0_32
OPEN #8,con_20x50
OPEN #7,con_20x50a10x10
```

SCR_*w***X***h***a***x***X***y*

Screen Output

[*w***X***h*] - window, *width, height*
[**A***x***X***y*] - window *X, Y* coordinate
default: scr_448x180a32x16

example:
```
OPEN #4, scr _0x10a20x50
OPEN #5, scr_10x10
```

SER*nphz*

Serial (RS-232-C)

n port number (1 or 2)

[*p*] parity	[*h*] handshaking	[*z*] protocol
e – even	**i** – ignore	**r** - raw data no EOF
o – odd	**h** – handshake	**z** - control Z is EOF
m – mark		**c** - as z but converts ASCII 10 (Qdos newline character) to ASCII 13 <CR>)
s – space		

default: **ser1rh** (8 bit no parity with handshake)

example:
```
OPEN #3, serle
OPEN #4, serc
COPY mdv1_test_file TO ser1c
```

NET*d_s*

Serial Network I/O

[*d*] indicates direction	[*s*] station number
i – input	**0** - for broadcast
o – output	**own station** - for general listen (input only)

default: no default

example:
```
OPEN #7, neti_32
OPEN #4, neto_0
COPY ser1 TO neto_21
```

MDVn_*name*

Microdrive File Access

n - Microdrive number
name - Microdrive file name

default: no default

example:
```
OPEN #9, mdv1_data_file
OPEN #9, mdv1_test_program
COPY mdv1_test_file TO scr_
```

Keyword	Function
OPEN	initialise a device and activate it for use
CLOSE	deactivate a device
COPY	copy data between devices
COPY_N	copy data between devices, but do not copy a file's header information
EOF	test for end of file
WIDTH	set width

DIRECT COMMAND

SuperBASIC makes a distinction between a statement typed in preceded by a line number and a statement typed in without a line nurnber. Without a line number the statement is a **direct command** and is processed immediately by the SuperBASIC **command interpreter**. For example, **RUN** is typed in on the command line and is processed, the effect being that the program starts to run. If a statement is typed in with a line number then the syntax of the line is checked and any detectable syntax errors reported. A correct line is entered into the SuperBASIC program and stored. These statements constitute a SuperBASIC *program* and will only be executed when the program is started with the **RUN** or **GOTO** command.

Not all SuperBASIC statements make sense when entered as a direct command, for example, **END FOR, END DEFine,** etc

ERROR HANDLING

Errors are reported by SuperBASIC in a standard form:

At line *line_number error_text*

Where the line number is the number of the line where the error was detected and the error text is listed below.

(1) Not complete
An operation has been prematurely terminated (or break has been pressed).

(2) Invalid job
An error return from Qdos relating to system calls controlling multitasking or I/O.

(3) Out of memory
Qdos and/or SuperBASIC has insufficient free memory.

(4) Out of range
Usually results from attempts to write outside a window or an incorrect array index.

(5) Buffer full
An I/O operation to fetch a buffer full of characters filled the buffer before a record terminator was found.

(6) Channel not open
Attempt to read, write or close a channel which has not been opened. Can also occur if an attempt to open a channel fails.

(7) Not found
File system, device, medium or file cannot be found. SuperBASIC cannot find an identifier. This can result from incorrectly nested structures.

(8) Already exists
The file system has found an already existing file with the same name as a new file to be opened for writing.

(9) In use
The file system has found that a file or device is already exclusively used.

(10) End of file
End of file detected during input.

(11) Drive full
A device has been filled (usually Microdrive).

(12) Bad name
The file system has recognised the name but there is a syntax or parameter value error. In SuperBASIC it means a name has been used out of context. For example, a variable has been used as a procedure.

(13) Xmit error
RS-232-C parity error

(14) Format failed
Attempted format operation has failed, the medium is possibly faulty (usually a Microdrive cartridge).

(15) Bad parameter
There is an error in the parameter list of a system or SuperBASIC procedure or function call. An attempt was made to read data from a write only device.

(16) Bad or changed medium
The medium (usually a Microdrive cartridge) is possibly faulty

(17) Error in expression
An error was detected while evaluating an expression.

(18) Overflow
Arithmetic overflow division by zero, square root of a negative number, etc.

(19) Not Implemented

(20) Read only
There has been an attempt to write data to a shared file.

(21) Bad line
A SuperBASIC syntax error has occurred.

(22) PROC/FN cleared
This is a message which is for information only and is not reporting an error. It is reporting that the program has been stopped and subsequently changed forcing SuperBASIC to reset its internal state to the outer program level and so losing any procedure environment which may have been in effect.

error recovery

After an error has occurred the program can be restarted at the nextstatement by typing

```
CONTINUE
```

If the error condition can be corrected, without changing the program, the program can be restarted at the statement which triggered the error. Type

```
RETRY
```

EXPRESSIONS

SuperBASIC expressions can be *string, numeric, logical* or a mixture: unsuitable data types are automatically converted to a suitable form by the system wherever this is possible.

define

monop: = | **+**
| **-**
| **NOT**

expression: = | *[monop] expression operator expression*
| *(expression)*
| *atom*

atom: = | *variable*
| *constant*
| *function* | *(expression *|, expression *)*

| *array_element*

variable: = | *identifier*
| *identifier* ***%***
| *identifier* ***$***

function: = | *identifier*
| *identifier* ***%***
| *identifier* ***$***

constant: = | *digit * [digit] **
| **[digit] *, *[digit]***
| **[digit] * |,| **[digit]** **E** **[digit]***

The final value returned by the evaluation of the expression can be integer giving an **integer_expression**, string giving a **string_expression** or floating point giving a **floating expression**. Often floating point and integer expressions are equivalent and the term **numeric_expression** is then used.

Logical operators can be included in an expression. If the specified operation is true then a one is returned as the value of the operation. If the operation is false then a zero is returned. Though logical operators can be used in any expression they are usually used in the expression part of an **IF** statement.

example:
i. `test_data + 23.3 + 5`
ii. `"abcdefghijklmnopqrstuvwxyz"(2 TO 4)`
iii. `32.1 * (colour = 1)`
iv. `count = -limit`

FILE TYPES

FILES

All I/O on the QL is to or from a *logical file*. Various file types exist.

data

SuperBASIC programs, text files. Created using **PRINT, SAVE**, accessed using **INPUT, INKEY$, LOAD** etc.

exec

An executable transient program. Saved using **SEXEC**, loaded using **EXEC, EXEC_W** etc.

code

Raw memory data, screen images, etc. Saved using **SBYTES**, loaded using **LBYTES**.

FUNCTIONS AND PROCEDURES

SuperBASIC *functions and procedures* are defined with the **DEFine FuNction** and **DEFine PROCedure** statements. A function is activated (or called) by typing its name at the appropriate point in a SuperBASIC expression. The function must be included in an expression because it is returning a value and the value must be used. A procedure is activated (or called) by typing its name as the first item
in a SuperBASIC statement.

Data can be passed into a function or procedure by appending a list of **actual parameters** after the function or procedure name. This list is compared to a similar list appended after te name of the function or procedure when it was defined. This second list is called the **formal parameters** of the function or
procedure. The formal parameters must be SuperBASIC variables. The actual parameters must be an *array*, an *array slice* or a SuperBASIC *expression* of which a single *variable* or constant is the simplest form.

Since the actual parameters are actual expressions, they must have an actual type associated with them. The formal parameters are merely used to indicate how the actual parameters must be processed and so have no type associated with them. The items in each list of parameters are paired off in order when the function or procedure is called and the formal parameters become equivalent to the actual parameters. There are three distinct ways of using parameters.

If the actual parameter is a single variable and if data is assigned to the formal parameter in the function or procedure then the data is also assigned to the corresponding actual parameter.

If the actual parameter is an expression then assigning data to the corresponding formal parameter will have no effect outside the procedure. Note that a variable can be turned into an expression by enclosing it within brackets.

if the actual parameter is a variable but has not previously been set then assigning data to the corresponding formal parameter will set the variable specified as the actual parameter.

Variables can be defined to be local to a function or procedure with the **LOCal** statement. Local variables have no effect on similarly named variables outside the function or procedure in which they are defned and so allow greater freedom in choosing sensible variable names without the risk of corrupting external variables.

A local variable is available to any inside function or procedure called from the procedure function in which it is declared to be local unless the function or procedure called contains a further local declaration of the same variable name.

Functions and procedures in SuperBASIC can be used recursively. That is a function or procedure can call itself either directly or indirectly.

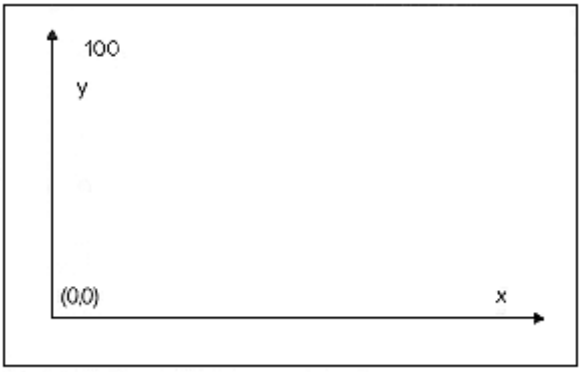
Command	Function
DEFine FuNction	define a function
DEFine PROCedure	define a procedure
RETurn	leave a function or procedure (return data from a function)
LOCal	define local data in a function or procedure

GRAPHICS

It is important to realise that the QL screen has non-square pixels and that changing screen mode will change the shape of the pixels. Thus if the grapics procedures were simply pixel based they would draw different shapes in the two modes. For example, in one mode we would have a circle while the same figure in the other mode would be an ellipse.

The graphics procedures ensure that whatever screen mode is in use, consistent figures are produced. It is not possible to use a simple pixel count to indicate sizes of figures, so instead the graphics procedures use an arbitrary scale and coordinate system to specify sizes and positions of figures.

The graphics procedures use the **graphics co-ordinate system**, i.e. draw relative to the **graphics origin** which is in the bottom left hand corner of the specified or default window. Note that this is not the same as the *pixel origin* used to define the position of *windows* and *blocks* etc. The graphics origin allows a standard Cartesian coordinate system to be used. A graphics cursor is updated after each graphics operation: subsequent operations can either be relative to this cursor or can be absolute, i.e. relative to the graphics origin.



The Graphics Coordinate System

The **scaling factor** is such that the full distance in the vertical direction in the specified or default window has length 100 by default and can be changed with the SCALE command. The scale in the x direction is equal to the scale in the y direction. However, the length of line which can be drawn in the x direction is dependent on the shape of the window. Increasing the scale factor increases the maximum size of the figure which can be drawn

before the window size is exceeded. If the graphics output is switched to a different size of window then the subsequent size of the output is adjusted to fit the new window. If the figure exceeds its output window then the figure is clipped.

It is useful to consider the window to be a window onto a larger graphics space in which the figures are drawn. The **SCALE** command allows the graphics origin to be set so allowing the window to be moved around the graphics space.

The graphics procedures are output to the window attached to the specified or default *channel* and the output is drawn in the **INK** colour for that channel.

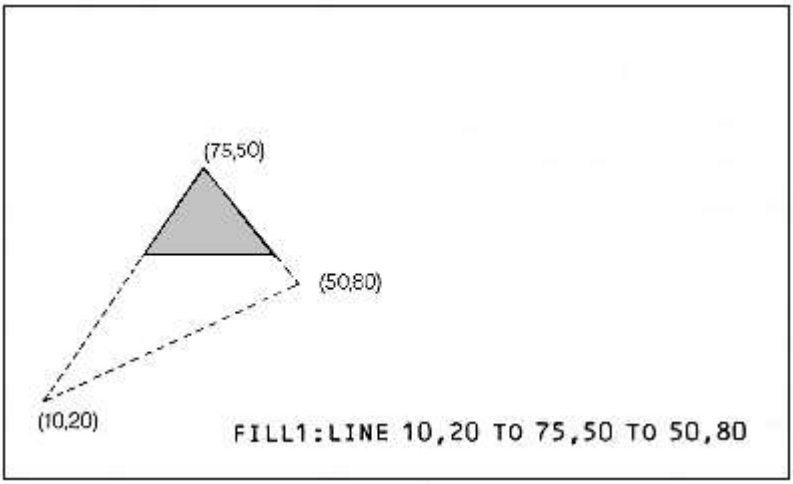
Command	Function	
CIRCLE	draw an ellipse or a circle	}
LINE	draw a line	} absolute
ARC	draw an arc of a circle	}
POINT	plot a point	}
CIRCLE_R	draw an ellipse or a circle	}
LINE_R	draw a line	}
ARC_R	draw an arc of a circle	} relative
POINT_R	plot a point	}
SCALE	set scale and move origin	
FILL	fill in a shape	
CURSOR	position text	

Graphics Fill

Figures drawn with the graphics and turtle graphics procedures can be optionally 'filled' with a specified stipple or colour. If **FILL** is selected then the figure is filled as it is drawn.

The **FILL** algorithm stores a list of points to plot rather than actually plotting them. When the figure closes there are two points on the same horizontal line. These two points are connected by a line in the current INK colour and the process repeats. Fill must always be reselected before drawing a new figure to ensure that the buffer used to store the list of points is reset.

The following diagram illustrates **FILL**:



warning

There is an implementation restriction on **FILL**. **FILL** must not be used for re-entrant shapes (i.e. a shape which is concave). Re-entrant shapes must be split into smaller shapes which are not re-entrant and each sub-shape filled independently.

IDENTIFIER

A SuperBASIC identifier is a sequence of letters, numbers and underscores.

define: letter:= | **a..Z**
| **A..Z**

number:= | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 |

identifier:= *letter* * || *letter* | *number* | _ | | *

example:
i. `a`
ii. `limit_1`
iii. `current_guess`
iv. `counter`

An identifier must begin with a letter followed by a sequence of letters, numbers and underscores and can be up to 255 characters long. Upper and lower case characters are equivalent.

Identifiers are used in the SuperBASIC system to identify Variables, *Procedures, Functions, Repetition* loops, etc.

warning

NO meaning can be attributed to an identifier other than its ability to identify constructs to SuperBASIC. SuperBASIC cannot infer the intended use of an identifier from the identifier's name!

JOYSTICK

The joystick ports marked **CTL1** and **CTL2**, allow two joysticks to be attached to the QL.

The joysticks are arranged to generate specific key depressions when moved in a specific way and any program which uses a joystick must be able to adapt to these keys. The QL keyboard can be read directly using the **KEYROW** function.

	CTL1	**CTL2**
mode	**key**	**key**
up	cursor up	**F4**
down	cursor down	**F2**
left	cursor left	**F1**
right	cursor right	**F3**
fire	space	**F5**

comment

The joystick ports can be used for adding other more general purpose control devices to the QL.

Joysticks for other computers using a 9-way 'D' connector require an adaptor to be used with the QL. Such an adaptor is available from Sinclair Research.

KEYWORD

SuperBASIC keywords are identifiers which are defined in the SuperBASIC Keyword Reference Guide. Keywords have the same form as a SuperBASIC standard identifier. The case of the keyword is not significant. Keywords are echoed as a mixture of upper and lower case letters and are always reproduced in full. The upper case portion indicates the minimum required to be typed in for SuperBASIC to recognise the keyword.

The set of SuperBASIC keywords may be extended by adding PROCEDURES to the QL. It is a good idea to define these with their names in upper case and this will indicate their special function in the SuperBASIC system. Conversely, ordinary procedures should be defined with their names in lower case.

WARNING: Existing keywords cannot be used as ordinary identifiers within a SuperBASIC program. SuperBASIC keywords are:

List of Keywords			
ABS	DEFine PROCedure	LEN	RANDOMISE
ACOS,ASIN	END DEFine	LET	RND
ACOT,ATAN	DEG	LIST	RECOL
ADATE	DELETE	LOAD	REMark
ARC,ARC_R	DIM	LOCal	RENUM
AT	DIMN	LN,LOG10	REPeat
AUTO	DIR	LRUN	END REPeat
BAUD	DIV	MERGE	RESPR
BEEP	DLINE	MOD	RETurn
BEEPING	EDIT	MODE	RETRY
BLOCK	ELLIPSE	MOVE	RUN
BORDER	ELLIPSE_R	MRUN	SAVE
CALL	EOF	NET	SIN
CHR$	EXEC,EXEC_W	NEW	SCALE
CIRCLE	EXIT	NEXT	SCROLL
CIRCLE_R	EXP	ON GO TO	SDATE
CLEAR	FILL	ON GO SUB	SELect
CLOSE	FILL$	OPEN,OPEN_IN	END SELect
CLS	FLASH	OPEN_NEW	SEXEC
CODE	FOR	OVER	SQRT
CONTINUE	END FOR	PAN	STOP
RETRY	FORMAT	PAPER	STRIP
COPY,COPY_N	GO SUB	PAUSE	TAN
COS	GO TO	PEEK,PEEK_W	TO
COT	IF,THEN,ELSE	PEEK_L	TURN
CSIZE	END IF	PENUP	TURNTO
CURSOR	INK	PENDOWN	UNDER
DATA,READ	INKEY$	PI	VER$
RESTORE	INPUT	POINT,POINT_R	WIDTH
DATE$,DATE	INSTR	POKE,POKE_W	WINDOW
DAY$	INT	POKE_L	
DEFine FuNction	KEYROW	PRINT	
END DEFine	LBYTES	RAD	

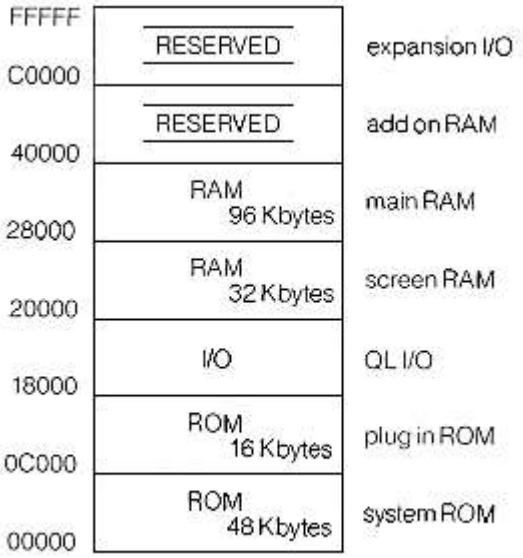
MATHS FUNCTIONS

SuperBASIC has the standard trigonometrical and mathematical functions.

Function	Name
COS	cosine
SIN	sin
TAN	tangent
ATAN	arctangent
ACOT	arcotangent
ACOS	arcosine
ASIN	arcsine
COT	cotangent
EXP	exponential
LN	natural logarithm
LOG10	common logarithm
INT	integer
ABS	absolute value
RAD	convert to radians
DEG	convert to degrees
PI	return the value of pi ±
RND	generate a random number
RANDOMISE	reseed the random number generator

MEMORY MAP

The QL contains a Motorola 68008 microprocessor, which can address 1 Megabyte of memory, i.e. from 00000 to FFFFF Hex. The use of addresses within this range are defined by Sinclair Research to be as follows:



Physical Memory Map

The screen RAM is organised as a series of sixteen bit words starting at address Hex 20000 and progressing in the order of the raster scan, i.e. from left to right with each display line and then from the top to the bottom of the picture. The bits within each word are organised so that a pixel to the left is always more significant than a pixel to the right (i.e. the pixel pattern on the screen looks the same as the binary pattern). However, the organisation of the colour information in the two screen modes is different:

high byte AO=0	low byte AO=1	mode
GGGGGGGG	RRRRRRRR	512 mode (high res)
GFGFGFGF	RBRBRBRB	256 mode (low res)

G—green B—blue R—red F—flash

Setting the Flash bit toggles the flash state and freezes the background colour for the flash to the value given by R, G and B for that pixel. Flashing is always reset at the beginning of each display line.

In high resolution mode, red and green specified together is interpreted by the hardware as white.

warning

Use of reserved areas in the memory map may cause incompatibility with future Sinclair products. Spurious output to addresses defined to be peripheral I/O addresses can cause

unpredictable behaviour. It is recommended that these areas are NOT written to and not used for any other purpose. Poking areas in use as Microdrive buffers can corrupt Microdrive data and can result in a loss of
information. Pokng areas in use such as system tables can cause the system to crash and can result in the loss of data and programs.

All I/O should be performed using either the relevant SuperBASIC commands or the QDOS Operating System traps.

MICRODRIVES

Microdrives provide the main permanent storage on the QL. Each Microdrive cartridge has a capacity of at least 100Kbytes. Available free memory space is allocated by QDOS as Microdrive buffers when necessary to improve performance.

Each blank cartridge must be **formatted** before use and can hold up to 255 sectors of 512 bytes per sector. QDOS keeps a directory of files stored on the cartridge. Each microdrive file is identified using a standard SuperBASIC file or device name.

A cartridge can be write protected be removing the small lug on the right hand side.

On receiving new blank microdrive cartridges, format them a few times to condition the tape.

general care

Physically each Microdrive cartridge contains a 200 inch loop of high quality video tape which is moved at 28 inches per second. The tape completes one circuit every 7.5 seconds.

NEVER touch the tape with your fingers or insert anything into the cartridge

NEVER turn the computer on or off with cartridges in place

ALWAYS store cartridges in their sleeves when not in use

ALWAYS insert or remove cartridges from the Microdrive slowly and carefully

ALWAYS ensure the cartridge is firmly installed before starting the microdrive

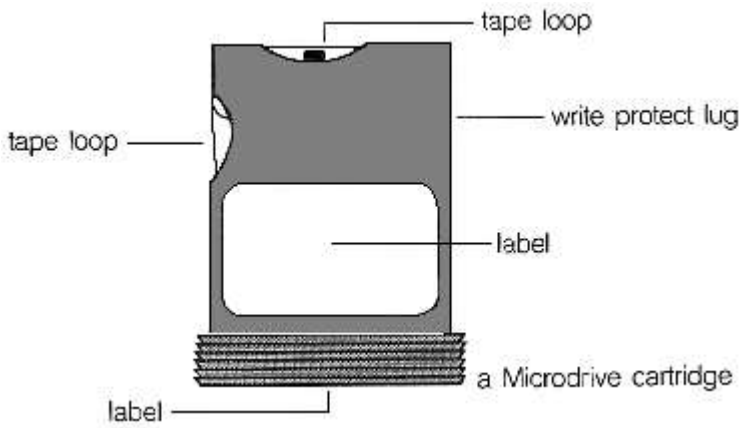
NEVER move the QL with cartridges installed - even if not in operation

NEVER touch the cartridge while the Microdrive is in operation

DO NOT repeatedly insert and remove the cartridge without running the Microdrive

tape loops

If a tape loop appears at either of the two places shown in figure 1 then gently ease it back into the cartridge. Use a non-fibrous instrument for this, e.g. the side of a pen or pencil. NEVER touch the tape with your fingers for this or any reason.



Command	Function
FORMAT	prepare a new cartridge for use
DELETE	delete a file from a cartridge
DIR	list the files on a cartridge
SAVE **SBYTES** **SEXEC**	saves data from a cartridge
LOAD **LBYTES** **EXEC** **MERGE**	loads data from a cartridge
OPEN_IN **OPEN_NEW** **OPEN** **CLOSE**	opens and closes files
PRINT **INPUT** **INKEY$**	SuperBASIC file I/O

warning

If you attempt to write to a cartridge which is write protected then the QL will repeatedly attempt to write the data but will eventually give up and give a "bad medium" error.

MONITOR

A monitor may be connected to the QL via the RGB socket on the back of the computer. Connection is via an 8-way DIN plug plus cable for colour monitors, or a 3-way DIN plug plus cable for monochrome. The RGB socket connections are as in the following table, and the column indicating wire colour refers to the colour coding used on the 8-way cable and connector available from Sinclair Research Limited. Pin designation is as shown in the diagram below.

Pin	function	signal		sleeve colour on QL RGB colour lead
1	PAL	composite PAL	(4)	orange
2	GND	ground		green
3	VIDEO	composite monochrome video	(3)	brown
4	CSYNC	composite sync	(2)	yellow
5	VSYNC	vertical sync	(1)	blue
6	GREEN	green	(1)	red
7	RED	red	(1)	white
8	BLUE	blue	(1)	purple

A monochrome monitor can be connected using a screened lead with a 3-way or an 8-way DIN plug at the QL end. Only pins 2 (ground) and 3 (composite video) need to be connected via the cable to the monitor. The connection at the monitor end will vary according to the monitor but is usually a phono plug. The monitor must have a 75 ohm 1V pk-pk composite video non-inverting input (which is the industry standard). Both 3-way DIN plugs and phono plugs are available from audio shops.

Diagram of Monitor Connector as viewed from rear of QL, showing pin numbers and functions.

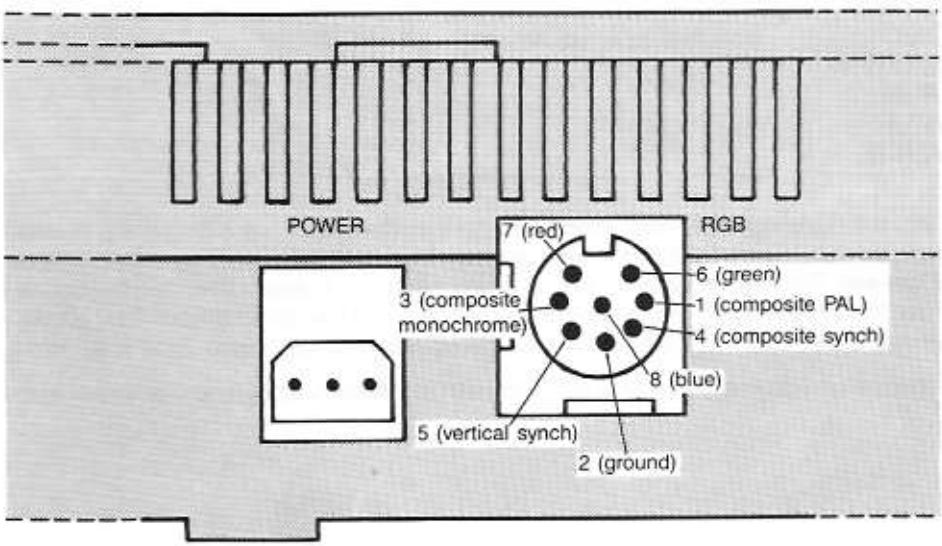


Diagram of Monitor Connector as viewed from rear of QL, showing pin numbers and functions

An RGB (colour) monitor can be connected using a lead with an 8 way DIN plug at the QL end. The connection at the monitor end will vary according to the monitor (there is no industry standard) and will often be supplied with it. A suitable cable with an 8-way DIN plug at one end and bare wires at the other end is available from Sinclair Research Limited.

A composite PAL monitor, or the composite video input on some VCRs **may** work with the QL. Only pins 2 (ground) and 1 (composite PAL) need to be connected via a cable to the monitor or VCR.

NETWORK

The QL can be connected with up to 63 other QLs. If there are more than 2 computers on the network then each computer (or station) must be assigned a unique station number. On the QL this can be done using the NET command.

Information is transmitted over the network in blocks. For normal communication between two stations the receiving station must acknowledge correct reception of the block. If a block is corrupted then the receiving station will request retransmission.

Using a network station number of zero has a special meaning. Sending to neto_0 is called broadcasting: any message sent in this way can be read by any station which is listening to neti_0. Note that the normal verification that a message has been received is disabled for broadcasts, so that broadcasting messages of length more than one block (255 bytes) is unreliable.

A network station which listens to its own station number (e.g. NET3:LOAD neti_3) can receive data from any station sending to it.

Command	Function
NET	assign a network station number
OPEN **CLOSE**	open a network channel close a network channel
PRINT **INPUT** **INKEY$**	network I/O
LOAD **SAVE** **LBYTES** **SBYTES** **EXEC** **SEXEC** **LRUN** **MRUN** **MERGE**	load and save via network

comment

If you are planning to connect several QLs on the network, or use a long piece of cable then you should wire it up with low capacitance twin core cable such as 3 amp light flex or bell wire. Take care to connect the centres of each jack to each other, and the outsides to each other. You will find that although the software can handle 63 stations, the hardware will not drive more than about 100m of cable, depending on what type it is.

If you are only connecting a few machines with the lads supplied, you need not worry.

OPERATORS

Operator	Type	Function
=	floating string	logical type 2 comparison
==	numeric string	almost equal ** (type 3 comparison)
+	numeric	addition
-	numeric	subtraction
/	numeric	division
*****	numeric	multiplication
<	numeric string	less than (type 2 comparison)
>	numeric string	greater than (type 2 comparison)
<=	numeric string	less than or equal to (type 2 comparison)
>=	numeric string	greater than or equal (type 2 comparison)
<>	numeric string	not equal to (type 3 comparison)
&	string	concatenation
&&	bitwise	AND
||	bitwise	OR

^^	bitwise	XOR
~	bitwise	NOT
OR	logical	OR
AND	logical	AND
XOR	logical	XOR
NOT	logical	NOT
MOD	integer	modulus
DIV	integer	divide
INSTR	string	type 1 string comparison
^	floating	raise to the power
-	floating	unary minus
+	floating	unary plus

**almost equal - equal to 1 part in 10^7

If the specified logical operation is true then a value not equal to zero will be returned. If the operation is false then a value of zero will be returned.

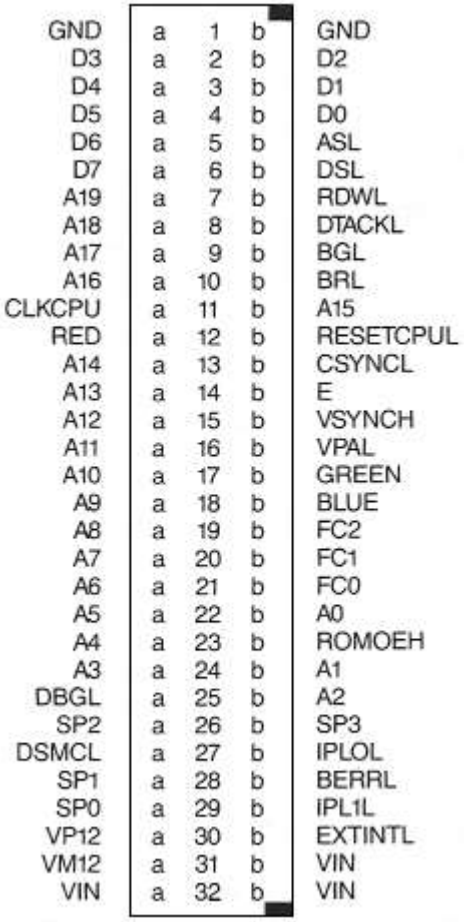
precedence

The precedence of SuperBASIC operators is defined in the table above. If the order of evaluation in an expression cannot be deduced from this table then the relevant operations are performed from left to right. The inbuilt precedence of SuperBASIC operators can be overriden by enclosing the relevant sections of the expression in parentheses.

highest	unary plus and minus
	string concatenation
	INSTR
	exponentiation
	multiply, divide, modulus and integer divide
	add and subtract
	logical comparison
	NOT (bitwise or logical)
	AND (bitwise or logical)
lowest	OR and XOR (bitwise or logical)

PERIPHERAL EXPANSION

The expansion connector allows extra peripherals to be plugged into the QL. The connections available at the connector are:



The connector on the QL is a 64 way (male) DIN-41612 indirect edge connector.

An 'L' appended to a signal name indicates that the signal is active low.

Signal	Function
A0-A19	68008 address lines
RDWL	Read / Write
ASL	Address Strobe
DSL	Data Strobe
BGL	Bus Grant
DSMCL	Data Strobe - Master Chip
CLKCPU	CPU Clock
E	6800 peripherals clock
RED	Red
BLUE	Blue
GREEN	Green
CSYNCL	Composite Sync

VSYNCH	Vertical Sync
ROMOEH	ROM Output Enable
FC0	Processor status
FC1	Processor status
FC2	Processor status
RESETCPUL	Reset CPU

QL Peripheral Output Signals

Signal	Function
DTACKL	Data acknowledge
BRL	Bus request
VPAL	Valid Peripheral Address
IPL0L	Interrupt Priority Level 5
IPL1L	Interrupt Priority Level 2
BERRL	Bus Error
EXTINTL	External Interrupt
DBGL	Data bus grab

QL Peripheral Input Signals

Signal	**Function**
D0..D7	Data Lines

QL Peripheral Bi-directional Signals

Signal	**Functional**
SP0..SP3	Select peripheral 0 to 3
VIN	9V DC (nominal) - 500mA max.
VM12	-12V
VP12	+12V
GND	ground

Miscellaneous

It is not intended that the following description of the QL peripheral expansion mechanism be sufficient to implement an actual expansion device, but rather be read to gain a basic understanding of the expansion mechanism.

Single or multiple peripherals may be added to the QL up to a maximum of 16 devices. A single peripheral can be plugged directly into the QL Expansion Slot while multiple peripherals must be plugged into the QL Expansion Module, which in turn is plugged into the QL Expansion Slot via a buffer card.

In this context the term 'device' also includes expansion memory. Although the areas of the QL memory map allocated to expansion memory are different from those allocate to expansion devices, the basic mechanism is the same. Only one expansion memory peripheral can be plugged into the QL at any one time. The address space allocated for peripheral expansion in the QL Physical memory map allows 16 Kbytes per peripheral. This area must contain the memory mapped I/O required for the driver and the code for the driver itself.

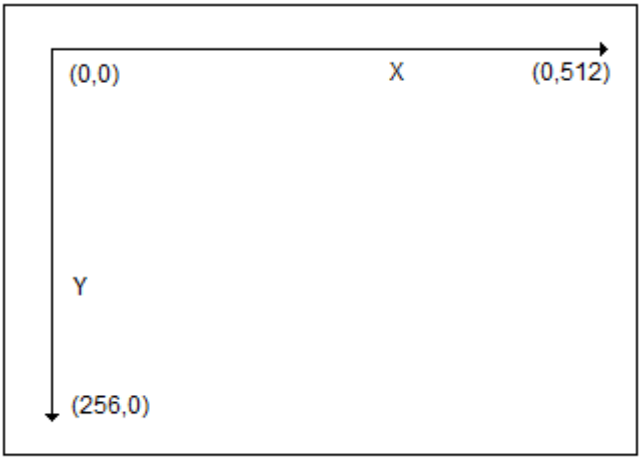
QDOS includes facilities for queue management and simple serial I/O which may be of use when writing device drivers.

The position of each peripheral device in the overall memory map of the QL is determined by the select peripheral lines: SP0, SP1, SP2 and SP3. These select lines generate a signal corresponding to the slot position in the QL expansion module, thus for a device to be selected the address input from address lines: A14, A15, A16 and A17 must be the same as the signals from SP0, SP1, SP2 and SP3
respectively.

PIXEL COORDINATE SYSTEM

The **pixel coordinate system** is used to define the positions and sizes of *windows*, **blocks** and **cursor** positions on the QL screen. The coordinate system has its origin in the top left hand corner of the default window (or screen) and always assumes that positions are specified as though the screen were in *512 mode* (high resolution mode). The system will use the nearest pixel available for the particular mode set making the coordinate system independent of the screen mode in use.

Some commands are always relative to the default window origin, e.g. **WINDOW**, while some are always relative to the current window origin, e.g. **BLOCK**



The Pixel Coordinate System

PROGRAM

A SuperBASIC program consists of a sequence of SuperBASIC *statements*, where each statement is preceded by a *line number*. Line numbers are in the range of 1 to 32767.

Command	Function
RUN	start a loaded program
LRUN	load a program from a device and start it
[CTRL] [SPACE]	force a program to stop

syntax: *line_number:= *[digit]* [range 1,32767]*
[line_number statement *[:statement]*]

example: i.

```
100 PRINT "This is a valid line number"
RUN
```

ii.

```
100 REMark a small program
110 FOR foreground = 0 TO 7
120   FOR contrast = 0 TO 7
130     FOR stipple = 0 TO 3
140       PAPER foreground, contrast, stipple
150       CURSOR 0,70
160       FOR n = 0 TO 2
170         SCROLL 2,1
180         SCROLL -2,2
190       END FOR n
200     END FOR stipple
210   END FOR contrast
220 END FOR foreground
RUN
```

QDOS

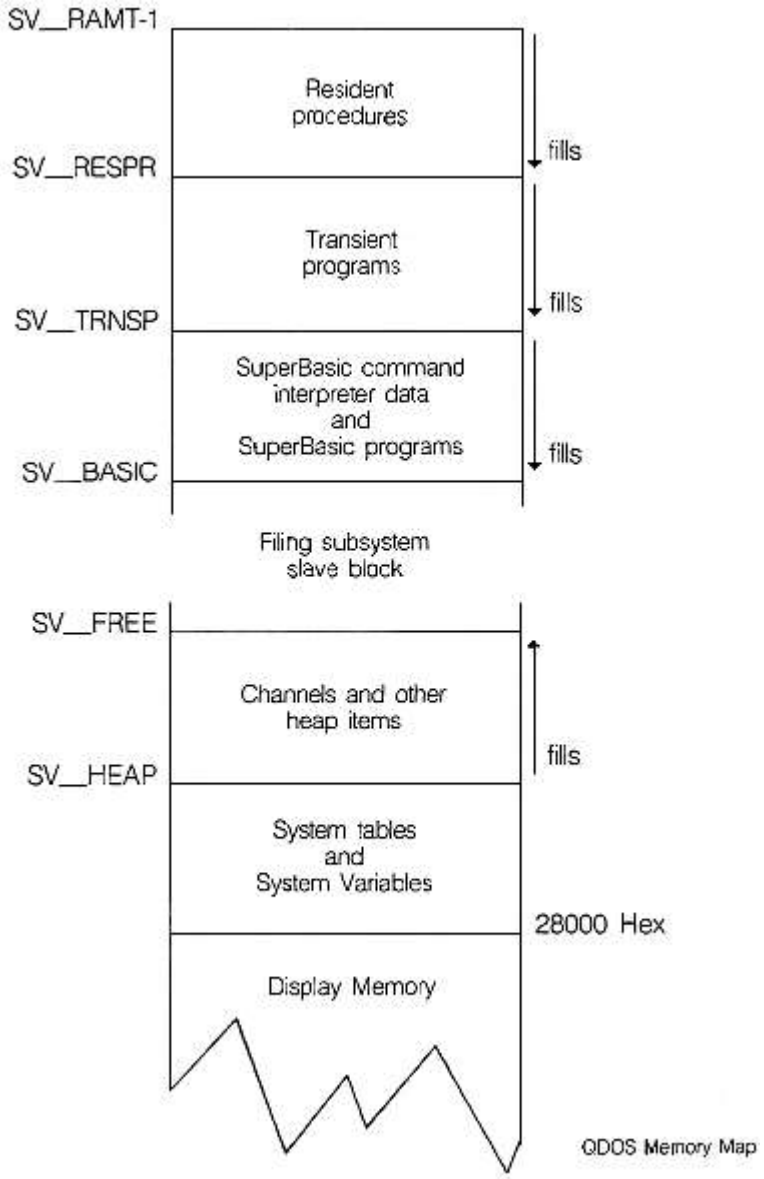
Qdos is the QL Operating System and supervises:

Task Scheduling and resource allocation
Screen I/O (including windowing)
Microdrive I/O
Network and serial channel communication
Keyboard input
Memory management

memory map

A full description of Qdos is beyond the scope of this guide but a brief description is included.

The system RAM has an organisation imposed by the QDOS operating system and is defined as follows:



QDOS Memory Map

The terms SV_RAMT, SV_RESPR, SV_TRNSP, SV_BASIC, SV_FREE, SV_HEAP are used to represent addresses inside the QL. These terms are not recognised by

SuperBASIC or the QDOS operating system. Furthermore, the addresses represented are liable to change as the system is running.

sv_ramt **RAM Top**
This will vary according to the memory expansion boards attached to the system.

sv_respr **Resident Procedures**
Resident procedures are loaded into the top of RAM. Space can be allocated in the resident procedure area using the RESPR function, but this space cannot be released except by resetting the QL. Resident Procedures written in machine code can be added to the SuperBASIC name list and so become extensions to the SuperBASIC system.

sv_trnsp **Transient Programs**
Transient programs are loaded immediately below the resident procedures. Each program must be self contained, i.e. it must contain space for its own data and its own stack. It must be position independent or must be loaded by a specially written linking loader. A transient program is executed from BASIC by using the **EXEC** command or from QDOS by activating it as a job.

The transient program area may be used for storing data only but this data will still be treated by QDOS as a job and therefore must not be activated.

sv_basic **SuperBASIC Area**
This area contains all loaded SuperBASIC programs and related data. This area expands and contracts using up the free space as required.

sv_free **Free Space**
Free space is used by the Qdos file subsystem to create Microdrive Slave Blocks, i.e. copies of Microdrive blocks which can be held in RAM.

sv_heap **System Heap**
This is used by the system to store data channel definitions and also provides working storage for the I/O subsystem. Transient programs may allocate working space for themselves on the heap via Qdos system calls.

System Tables/System Variables
This area is directly above the screen memory. The System Tables and supervisor stack are resident above the system variables.

system calls

System calls are processed by Qdos in 'supervisor mode'. When in supervisor mode, Qdos will not allow any other job to take over the processor. System calls processed in this way are said to be 'atomic', i.e. the system call will process to completion before relinquishing the processor. Some system calls are only partially atomic, i.e. once they have completed their primary function they will relinquish the processor if necessary. Unless specifically requested all the system calls are partially atomic.

The standard mechanism for making a system call is by making a trap to one of the Qdos system vectors with appropriate parameters in the processor registers. The action taken by Qdos following a system call is dependent on the particular call and the overall state of the system at the time the call was made.

input/output

Qdos supports a multitasking environemtn and therefore a file can be accessed by more than one process at a time. The Qdos filing sub-system can handle files which have been opened as EXCLUSIVE files or as SHARED files. A shared file cannot be written to. QL devices are processed by the SERIAL I/O SYSTEM. As its name suggests any data output by this system can be redirected to any other
device also supported by the redirectable I/O system.

The device names required by Qdos are the same as the device names required by SuperBASIC and are discussed in the concept section DEVICES. The collection of standard devices supplied with the QL can be expanded.

devices

The standard devices included in the system are discussed in this guide in the section DEVICES. Further devices may be added to the system, given a name (e.g. SER1, NET) and then accessed in the same way as any other QL device.

multitasking

Jobs will be allowed a share of the CPU in line with their priority and competition with other jobs in the system. Jobs running under the control of Qdos can be in one of three states:

active: Capable of running and sharing system resources. A job in this state may not be runnign continuously but will obtain a share of the CPU in line with its priority.

suspended: The job is capable of running but is waiting for another job or I/O. A job may be suspended indefinitely or for a specific period of time.

inactive: The job is incapable of running, its priority is 0 and so it can never obtain a share of the CPU

Qdos will reschedule the system automatically at a rate related to the 50 Hz frame rate. The system will also be rescheduled after certain system calls.

example: This program generates an on-screen readout of the real-time clock running as an independent job.

First **RUN** this program with a formatted cartridge in microdrive 2. This generates a machine code title called 'clock'. Wait for the microdrive to stop. Next, set the clock using the **SDATE** command.

Then type:

```
EXEC mdv2_clock
```

and a continuous time display will appear at the top right of the command window.

```
100  c=RESPR(100)
110  FOR i = 0 TO 68 STEP 2
120    READ x:POKE_W i+c,x
130  END FOR i
140  SEXEC mdv2_clock,c,100,256
1000 DATA 29439,29697,28683,20033,17402
1010 DATA 48,13944,200,20115,12040
1020 DATA 28691,20033,17402,74,-27698
1030 DATA 13944,236,20115,8279,-11314
1040 DATA 13944,208,20115,16961,16962
1050 DATA 30463,28688,20035,24794
1060 DATA 0,7,240,10,272,200
```

N.B. Line 1060 governs the position and colour of the clock window - the data terms are, in order:

border colour/width, paper/ink colour, window width, height, x-origin, y-origin

These are pairs of bytes, entered by **POKE_W** as words.

The x-origin and the y-origin (the last data item) should be 272 and 202 in monitor mode, or 240 and 216 in TV mode.

Generate the paper and ink word, for example, as 256*paper+ink. Thus white paper, red ink is 256*7 + 2 = 1794

REPETITION

Repetition in SuperBASIC is controlled by two basic program constructs. Each construct must be identified to SuperBASIC:

REPeat *identifier*	**FOR** *identifier* = *range*
Statements	*statements*
END REPeat *identifier*	**END FOR** *identifier*

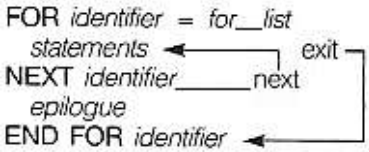
These two constructs are used in conjunction with two other SuperBASIC statements:

NEXT *identifier*	**EXIT** *identifier*

Processing a **NEXT** statement will either pass control to the statement following the appropriate **FOR** or **REPeat** statement, or if a **FOR** range has been exhausted to the statement following the **NEXT**.

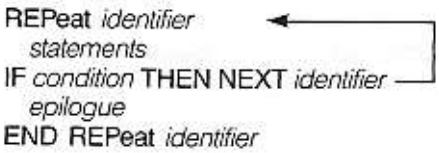
Prcoessing an **EXIT** will pass control to the statement after the **END FOR** or **END REPeat** selected by the **EXIT** statement. **EXIT** can be used to exit through many levels of nested repeat structures. **EXIT** should always be used in **REPeat** loops to terminate the loop on some condition.

A combination of **NEXT,EXIT** and **END** statements allows **FOR** and **REPeat** loops to have a **loop epilogue** added. A loop epilogue is a series of SuperBASIC statements which are executed on some special condition arising within the loop:



The loop epilogue is only processed if the **FOR** loop terminates normally. If the loop terminates via an **EXIT** statement then processing will continue at the **END FOR** and the epilogue will not be processed.

It is possible to have a similar construction in a **REPeat** loop:



This time entry into the loop epilogue is controlled by the **IF** statement. The epilogue will or will not be processed depending on the condition in the **IF** statement. A **SELect** statement can also be used to control entry into the epilogue.

ROM CARTRIDGE SLOT

Allows software to be used in the QL system from a Sinclair QL ROM Cartridge. The ROM Cartrdge can contain software to directly change the behaviour of the SuperBASIC system. The cartridge can contain:

i. Software to be used instead of or with the SuperBASIC system. For example:

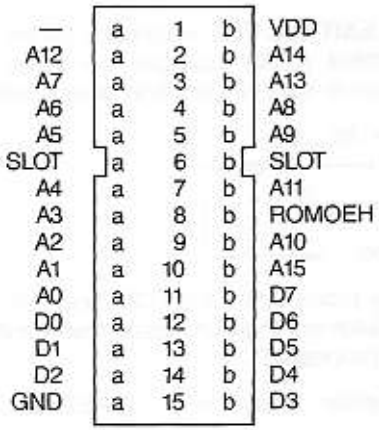
assemblers
compilers
debuggers
application software
etc

ii. Software to expand the SuperBASIC system. For example:

special procedures
etc

It is not possible to use ZX ROM Cartridges on the QL.

pin out



Side b is the upper side of the connector; side a is the lower.

Signal	Function
A0..A15	Address lines
D0..D7	Data lines
ROMOEH	ROM Output Enable
VDD	5V
GND	Ground

warning:

Never plug or unplug a ROM cartridge while the QL power is on.

SCREEN

512 mode

The screen is 512 pixels across and 256 pixels deep. Only the solid colours

black
red
green
white

can be displayed in this mode.

256 mode

Low resolution mode also has a hardware flash. The screen is 256 pixels across and 256 pixels deep. The full set of solid colours is available in this mode:

black
blue
red
magenta
green
cyan
yellow
white

warning

A domestic television is not capable of displaying the complete QL screen. Portions of the screen at the top and the sides will not be reproduced. The default initial window will take account of this and will reduce the effective picture size. The full size can be restored with the **WINDOW** command.

Command	Function
MODE	set screen mode

SLICING

Under certain circumstances it is possible to refer to more than one element in an array i.e. slice the array The array slice can be thought of as defining a **subarray** or a series of

subarrays to SuperBASIC. Each slice can define a continuous sequence of elements belonging to a particular dimension of the
original array. The term array in this context can include a numeric array, a string array or a simple string.

It is not necessary to specify an index for the full number of dimensions of an array. If a dimension is omitted then slices are added which will select the full range of elements for that particular dimension, i.e. the slice (0 TO). SuperBASIC can only add slices to the end of a list of array indices.

syntax: *index: =* | *numeric_exp* {single element}
| *numeric_exp* **TO** *numeric_exp* {range of elements}
| *numeric_exp* **TO** {range to end}
| **TO** *numeric_expression* {range from beginning}

array_reference:= | *variable*
| *variable (* | *index* * |*,index*| * | *)*

An array slice can be used to specify a source or a destination subarray for an assignment statement.

example: i. `PRINT data array`
ii. `PRINT letters$(1 TO 15)`
iii. `PRINT two_d_array (3) (2 TO 4)`

String slicing is performed in the same way as slicing numeric or string arrays.

Thus

a$(n) will select the nth character.
a$(n TO m) will select all characters from the nth to the mth, inclusively
a$(n TO) will select from a character n to the end, inclusively
a$(1 TO m) will select from the beginning to the nth character inclusively
**a$** will select the entire contents of a a$

Some forms of **BASIC** have functions called **LEFT$, MID$, RIGHTS.** These are not necessary in SuperBASIC. Their equivalents are specified below:

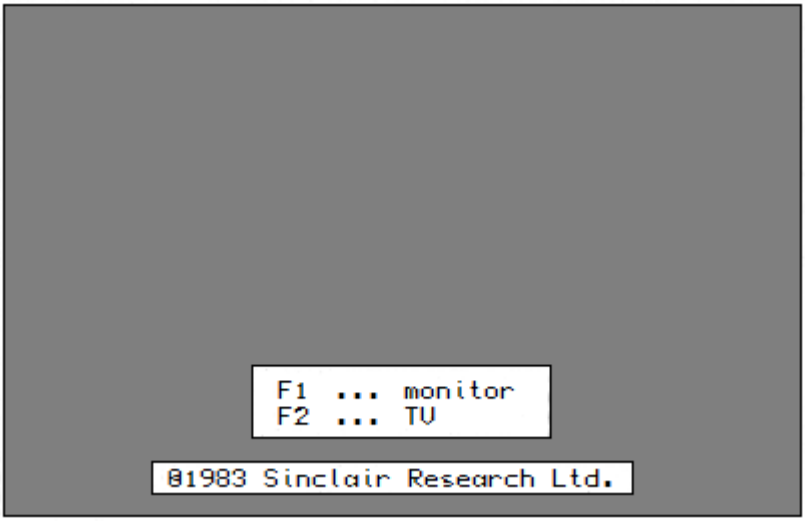
SuperBASIC	Other BASIC
a$(n)	MID$(a$,n,1)
a$(n TO m)	MID$ (a$,n,m+1-n)
a$(1 TO n)	LEFT$ (a$,n)
a$(n TO)	RIGHTS (a$,LEN(a$)+1-n)

warning

Assigning data to a sliced string array or string variable may not have the desired effect. Assignments made in this way will not update the length of the string. The length of a string array or string variable is only updated when an assignment is made to the whole string.

START UP

Immediately after switch on (or reset) the QL will perform a RAM test which will give a spurious pattern on the display. If the RAM test is passed then the screen will be cleared and the copyright screen displayed.



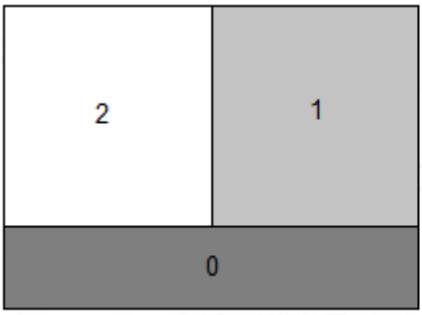
After start up, the QL displays the copyright message and asks whether it is being used on a television or a monitor. The QL will set different initial screen modes and window sizes depending on the answer.

Press F1 if you are using a monitor and F2 if you are using a television set.

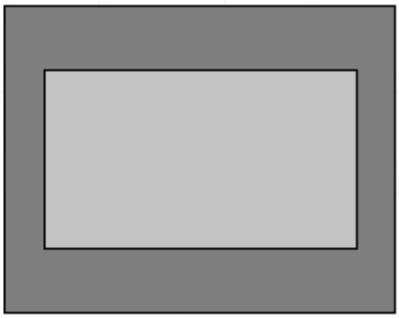
The QL has the ability to 'boot' itself up from programs contained in either the ROM cartridge slot or in Microdrive 1. If the ROM cartridge slot contains a self starting program then start up will continue under the control of the program in the ROM cartridge. If nothing suitable is found then the QL will check Microdrive 1 for a cartridge. If a cartridge is found and if it contains a file called BOOT it is loaded and run.

default screen

The QL has three default channels which are linked to three default windows.



Monitor


Television

Channel 0 is used for listing commands and error messages, channel 1 for program and graphics output and channel 2 for program listings. The default channel can be modified using the optional channel specifier in the relevant command.

It is important NOT to switch on the QL with a Microdrive cartridge in position. If booting from a Microdrive cartridge is required then the cartridge must be inserted between switching on and pressing either F1 or F2.

SOUND

Sound on the QL is generated by the QL's second processor (an 8049) and is controlled by specifying:

up to two pitches

the rate at which the sound must move between the pitches, the ramp

how the sound is to behave after it has reached one of the specified pitches, the wrap

if any randomness should be built into the sound, i.e. deviations from the ramp

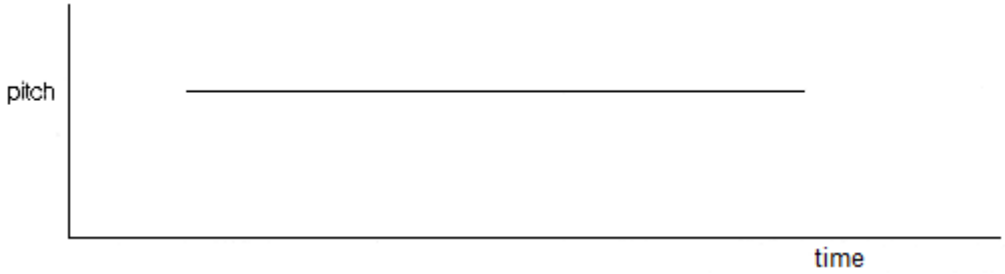
if any fuzziness should be built into the sound. i.e. deviations on every cycle of the sound

Fuzziness tends to result in buzzy sounds while randomness, depending on the other parameters, will result in 'melodic' sounds or noise.

The complexity of the sound can be built up stage by stage gradually building more complex sounds. This is, in fact, the best way to master sound on the QL.

Specify a duration and a single pitch. The specified pitch will be beeped for the specified time.

LEVEL 1

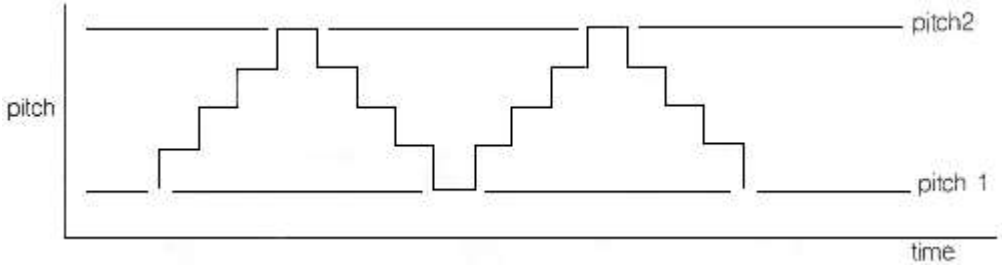


This is the simplest sound command, other than the command to stop the sound, on the QL.

LEVEL 2

A second pitch and a gradient can be added to the command. The sound will then 'bounce' between the two pitches at the rate specified by the gradient.

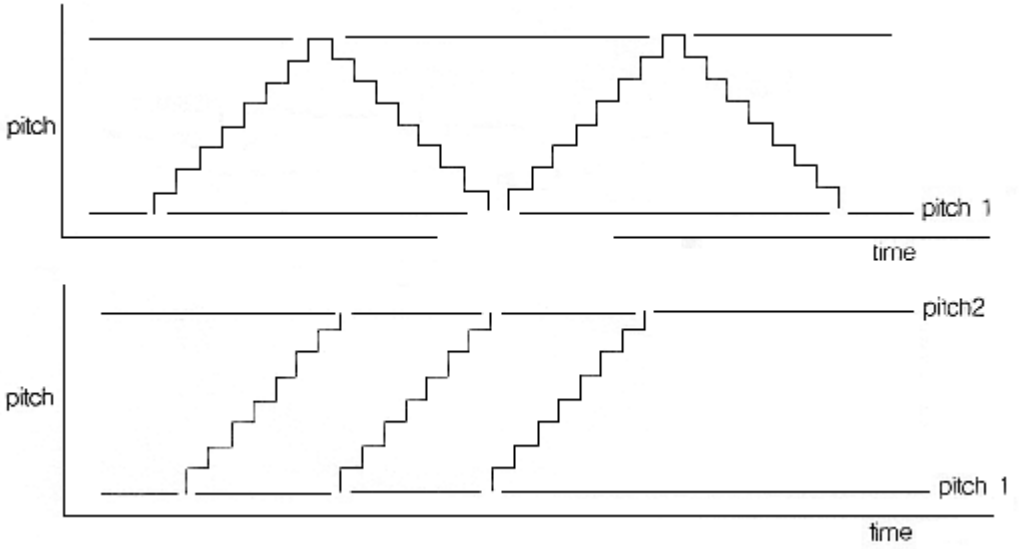
The sounds produced at this level can vary between: semi musical beeps, growls, zaps and moans. It is best to experiment.



LEVEL3

A parameter can be added which controls how the sound behaves when it becomes equal to one of the specified pitches. The sound can be made to 'bounce' or 'wrap'.

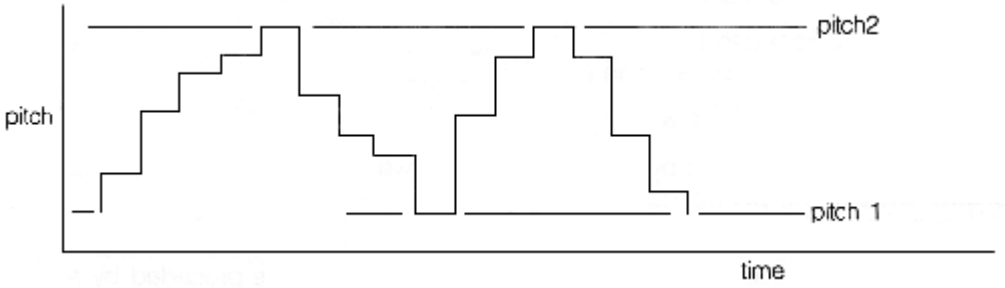
The number of wraps can be specified, including wrap forever. It is even more important to experiment.



LEVEL4

Randomness can be added to the sound. This is a deviation from the specified step or gradient.

Depending on the amount of randomness added in relation to the pitches and the gradient, it will generate a very wide and unexpected range of sounds.



LEVEL 5

More variation can be added by specifying 'fuzziness'. Fuzziness adds a random factor to the pitch continuously Fuzziness tends to make the sound buzz.

Combining all of the above effects can make a very wide range of sounds, many of them unexpected. QL sound is best explored through experiment. By specifying a time interval of zero the sound can be made to repeat forever and so a sequence of **BEEP** commands can be used until the sound generated is the sound which is required. A word of warning: slight changes in the value of a single parameter can have alarming results on the sound generated.

STATEMENT

A SuperBASIC statement is an instruction to the QL to perform a specific operation, for example:

```
LET a = 2
```

will assign the value 2 to the variable identified by **a**.

More than one statement can be written on a single line by separating the individual statements from each other by a colon (:), for example:

```
LET a = a + 2 : PRINT a
```

will add 2 to the value identified by the variable a and will store the result back in a. The answer will then be printed out

If a line is not preceded by a line number then the line is a direct command and SuperBASIC processes the statement immediately. If the statement is preceded by a line number then the statement becomes part of a SuperBASIC program and is added into the SuperBASIC program area for later execution.

Certain SuperBASIC statements can have an effect on the other statements over the rest of the logical line in which they appear i.e. **IF, FOR, REPeat, REM**, etc. It is meaningless to use certain SuperBASIC statements as direct commands.

STRING ARRAYS, STRING VARIABLES

String arrays and numeric arrays are essentially the same, however there are slight differences in treatment by SuperBASIC. The last dimension of a string array defines the maximum length of the strings within the array. String variables can be any length up to 32766. Both string arrays and string
variables can be sliced.

String lengths on either side of a string assignment need not be equal. If the sizes are not the same then either the right hand string is truncated to fit or the length of the left hand string is reduced to match. If an assignment is made to a sliced string then if necessary the 'hole' defined by the slice will be padded with spaces.

It is not necessary to specify the final dimension of a string array. Not specifying the dimension selects the whole string while specifying a single element will pick out a single character and specifying a slice will define a sub string.

COMMENT: Unlike many BASICs SuperBASIC does not treat string arrays as fixed length strings. If the data stored in a string array is less than the maximum size of the string array then the length of the string is reduced.

WARNING: Assigning data to a sliced string array Or string variable may not have the desired effect. Assignments made in this way will not update the length of the string and so it is possible that the system will not recognise the assignment. The length of a string array or a string variable is only
updated when an assignment is made to the whole string.

Command	Function
FILL$	generate a string
LEN	find the length of a string

STRING COMPARISON

order:

. (decimal point/full stop)
digits or numbers in numerical order
AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz
space ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [|] ^ _ / { | } ~ ©
other non printing characters

The relationship of one string to another may be:

equal: All characters or numbers are the same or equivalent

lesser: The first part of the string, which is different from the corresponding character in the second string, is before it in the defined order.

greater: Thefirst part of the first string which is different from the corresponding character in the second string, is after it in the defined order.

Note that a '.' may be treated as a decimal point in the case of string comparison which sorts numbers (such as SuperBASIC comparisons). Note also that comparison of strings containing non-printable characters may give unexpected results.

types of comparison

type 0 case dependent - character by character comparison

type 1 case independent - character by character

type 2 case dependent - numbers are sorted in numerical order

type 3 case independent - numbers are sorted in numerical order

type 0 not normally used by the SuperBASIC system.

usage

type 1 File and variable comparison
type 2 SuperBASIC <, <=, =, >= ,>, INSTR and <>
type 3 SuperBASIC == (equivalence)

SYNTAX DEFINITIONS

SuperBASlC syntax is defined using a non-rigorous 'meta language' type notation. Four types of construction are used :

| | Select one of
[ ] Enclosed item(s) are optional
* * Enclosed items are repeated

.. Range

{ } Comment

e.g.	| A | B |	A or B
	[A]	A is optional
	* A *	A is repeated
	A..Z	A, B, C, etc
	{this is a comment}	

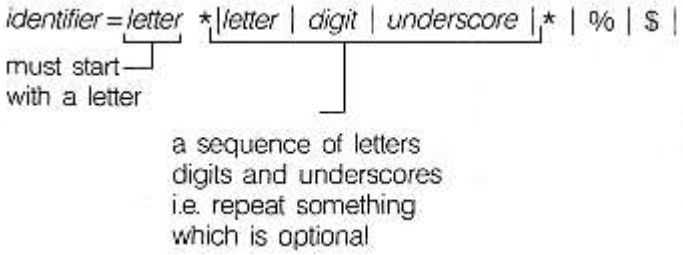
Consider a SuperBASIC identifier.

A sequence of numbers, digits, underscores, starting with a letter and finishing with an optional % or $

letter:= | **A..Z**
| **a..z**
{a letter is one of: ABCDEFGHIJKLMNOPQRSTUVWXYZ}
or abcdefghijklmnopqrstuvwxyz

digit: = | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Underscore:= _
{an underscore is _}



TURTLE GRAPHICS

SuperBASIC has a set of turtle graphics commands:

Command	Function
PENUP	stop drawing
PENDOWN	start drawing
MOVE	move the turtle
TURN	turn the turtle
TURNTO	turn to a specific heading

The set of commands is the minimum and normally would be used within another procedure to expand on the commands. For example:

```
100 DEFine PROCedure forward(distance)
110    MOVE distance
120 END DEFine
130 DEFine PROCedure backwards(distance)
140    MOVE -distance
150 END DEFine
160 DEFine PROCedure left(angle)
170    TURN angle
180 END DEFine
190 DEFine PROCedure right(angle)
200    TURN -angle
210 END DEFine
```

These will define some of the more famous turtle graphic commands.

Initially the turtle's pen is up and the turtle is pointing at 0 degrees which is to the right hand side of the window.

The FILL command will also work with figures drawn with turtle graphics. Also ordinary graphics and turtle graphics can be mixed, although the direction of the turtle is not modified by the ordinary graphics commands.

WINDOWS

Windows are areas of the screen which behave, in most respects, as though each individual window was a screen in its own right, i.e. the window will scroll when it has become filled by text, it can be cleared with the CLS command, etc.

Windows can be specified and linked to a channel when the channel is opened. The current window shape can be changed with the WINDOW command and a border added to a window with the BORDER command. Output can be directed to a window by printing to the relevant channel. Input can be directed to have come from a particular window by inputting from the relevant channel If more than one
channel is ready for input then input can be switched between the ready channels by pressing

[CTRL] C

The cursor will flash in the selected window

Windows can be used for graphics and non-graphic output at the same time. The non graphic output is relative to the current cursor position which can be positioned anywhere within the specified window with the CURSOR command and at any line-column boundary with the AT command. The graphics output is relative to a graphics cursor which can be positioned and manipulated with the graphics procedures.

PARTS

Certain commands (CLS, PAN etc.) will accept an optional parameter to define part of the current window for their operation. This parameter is as defined below:

part	description
0	whole screen
1	above and excluding cursor line
2	bottom of screen excluding cursor line
3	whole of cursor line
4	line right of and including cursor

Command	Function
WINDOW	re-define a window
BORDER	take a border from a window
PAPER	define the paper colour for a window
INK	define the ink colour for a window
STRIP	define a strip colour for a window
PAN	pan a window's contents
SCROLL	scroll a window's contents
AT	position the print position
CLS	clear a window
CSIZE	set character size
FLASH	character flash
RECOL	recolour a window