

QLiberator Technical Notes

15 Jun 2025

Table of Contents

The runtime module.....	2
Compiled jobs data area.....	3
Fake SuperBASIC variables area, stored at the start of the jobs data area.....	4
Runtime pointers.....	5
Runtime code processing.....	6
Variables area.....	7
Possible bug in runtimes.....	8
Compiled jobs A7 stack on startup.....	9
Code which does not seem to be ever called in the runtimes.....	10
QLIB_OBJ.....	11
Compiler data block.....	11
Tokenized program line buffer.....	11
QLIB_OVL.....	12
Overlay table.....	12
The OVERLAY command creates two new name table types.....	12
possible bug in UNLOAD command.....	12
QLIB_BIN.....	13
No label to message.....	13
Buffer to be the work file??.....	13
QLIB_EXT.....	14
Uncalled code.....	14
Q_ERR.....	14
QLIB1_BIN.....	15
QLIB_OBJ support files.....	15
QLIB2_BIN.....	17
QLIB3_BIN.....	19
QLIB4_BIN.....	19
QLIB5_BIN.....	20
QLIB6_BIN.....	20
QLIB7_BIN/CONFIG_BIN.....	21
Layout of compiled object file.....	22
Header area.....	23
Initialization and Runtime module.....	23
Compiled SuperBASIC program.....	23
SuperBASIC Extensions (optional).....	24
Jump for Externals.....	24
Line number list.....	24
Name list.....	24
Externals list.....	25
Name/Variable table.....	26

The runtime module

The runtime module has various entry points

base	Initializes the runtime module and adds a replacement CALL command
base + 4	Writes out the initialization message
base + 8	Main entry point from a _obj file. Initializes the jobs data area and starts the compiled SuperBASIC program
base + 16	Do Externals, D4 is an offset from the start of the compiled BASIC to the start of the proc/fun definition (code 76)
base + 20	Overlays, called from QLIB_OVL

There are also some pointers to parts of the runtime module, relative to the base address of the runtimes

base + 12	A word offset relative to the start of the runtimes to the code to continue a compiled program???
base + 14	A word containing the length of the runtime module
base + 24	A word offset relative to the start of the runtimes to the start up window definitions

If the runtimes have been loaded into memory by an **LRESPR**, then the start address of the runtimes can be found by **PEEK_L(\\\$EC)**. And the initialization message can be seen with the **Q_L** command.

Compiled jobs data area

The runtime module creates a SuperBASIC like memory map in the compiled jobs data area

Data area end

A7 stack 512 bytes	
Maths stack	sb_arthb/bv_ribas sb_arthp/bv_rip
Free space for maths stack expansion	lv_frbas
Stack stksz bytes	
Heap heapsz bytes	lv_hpbas
Variables area	lv_vvend
Name table	sb_nmtbb/bv_ntbas/lv_ntbas
Runtime pointers 24 bytes	
Return stack retstk bytes	sb_retsb/bv_rtbas
Channel table chcnt * \$28 bytes	sb_chanp/bv_chp sb_chanb/bv_chbas
SB Buffer buffsz bytes	lv_bftop sb_buffb/bv_bfbas
SuperBASIC variables 256 bytes	bv_end

Data area start

Fake SuperBASIC variables area, stored at the start of the jobs data area

bv_bfbas	sb_buffb	\$00	long	buffer base
bv_bfp	sb_buffp	\$04	long	buffer running pointer
lv_bftop	sb_cmdlb	\$08	long	top of buffer
lv_sbdat	sb_cmdlp	\$0c	long	address of SuperBASIC DATA area
		\$10	long	liberate
		\$14	long	liberate
bv_ntbas	sb_nmtbb	\$18	long	name table
bv_ntp	sb_nmtbp	\$1c	long	
bv_nlb	sb_nmlsb	\$20	long	name list
bv_nlp	sb_nmlsp	\$24	long	
bv_vvbas	sb_datab	\$28	long	variable values
bv_vvp		\$2c	long	variable values
bv_chbas	sb_chanb	\$30	long	channel table
bv_chp	sb_chanp	\$34	long	
bv_rtbas	sb_retsb	\$38	long	return table
bv_rtp	sb_retsp	\$3c	long	
lv_tgbas		\$54	long	zero in a compiled program
bv_rip	sb_arthp	\$58	long	relative arithmetic stack pointer
bv_ribas	sb_arthb	\$5c	long	relative arithmetic stack base (fills down)
lv_line		\$6c	byte	current line number
		\$6f	byte	liberate command
bv_rand	sb_rand	\$80	long	random number
lv_d6err		\$88	byte	latest D6 error code byte
lv_d6ler		\$89	byte	last/current error code??? initially 0
lv_jbsts		\$8a	byte	job status, \$FF job is suspended, \$00 job running
bv_brk	sb_break	\$8f	byte	0 if there has been a break, otherwise \$80
lv_frbas		\$90	long	address of start of free space, maths stack max
lv_ntbas		\$94	long	base of name table (absolute)
lv_basrp		\$98	long	running pointer of compiled SuperBASIC
lv_vvend		\$9c	long	end address of variables area
lv_sktop		\$a0	long	absolute address of base of maths stack (fills down)
lv_basic		\$a4	long	start address of compiled SuperBASIC
lv_hpbas		\$a8	long	base of heap
		\$ac	long	never written to??
		\$b0	long	never written to??
lv_rtadd		\$b4	long	return stack RETRY address
lv_qerno		\$c2	word	Q_ERR error number
lv_ernum		\$c4	word	ERNUM error number
lv_erlin		\$c6	word	ERLIN error line

lv_rtsav		\$ca	long	saved return stack pointer
lv_hptot		\$ce	long	total number of common heap requests
lv_scbuf		\$d0	long	address of a screen buffer, 0 = none, -1 = invalid
lv_hpreq		\$d4	long	total number space requested from the common heap
lv_jbext		\$d8	byte	some flag for an external job 0= , \$FF=
lv_cstr		\$d9	byte	flag for the type of string compare used in the system
lv_flag1		\$da	byte	liberate command 0= , \$FF=
lv_flag2		\$db	byte	liberate command 0= , \$FF=
lv_ertab	sb_qlibe	\$dc	long	Q_ERR error table base address
lv_cdblk	sb_cheap	\$e0	long	address of QLIB_OBJ compiler data block
lv_a7var		\$e4	long	address of a number of values on the A7 stack
lv_olbas	sb_qlibc	\$e8	long	base address of overlay area (128 bytes)
lv_rtime	sb_qlibr	\$ec	long	address of start of runtimes for the compiled job
lv_dprog		\$f0	long	address of compiled program pointer (DATA)
lv_a7ptr		\$f4	long	address of the A7 register, A7 storage
lv_rtp		\$f8	long	return stack pointer
lv_jbase		\$fc	long	address of start of job

Runtime pointers

This is an area of 24 bytes, just before the start of the name table. This area is usually accessed as a negative offset from the start of the name table.

Offset from
name table

-\$18	\$00	long	something to do with WHEN
-\$14	\$04	long	
-\$10	\$08	long	address of a name table
-\$0C	\$0c	long	start address of compiled job
-\$08	\$10	long	address of start of variables area
-\$04	\$14	long	address of a name list

Start of name table

Runtime code processing

During the main loop of processing command codes, When a code routine is entered. The registers are set as follows -

- D3 The code number
- A1 Points at the maths stack
- A2 Points at the base of the name table
- A3 ?
- A4 Compiled program pointer
- A5 Points at the 'Type' stack
- A6 Points at the base of the 'Fake' SuperBASIC area

Exit of the code routine -

- D6 Contains an error code if something went wrong
 - 1 to 28 (not 4) triggers an error message, anything greater triggers an 'Internal' error
 - 0
 - negative, triggers a runtime error

Variables area

Integers are stored as 2 bytes

Floats are stored as 6 bytes

FOR loop variable storage layout in the variables area is 24 bytes, for both integer and float.

Floating point FOR

\$0000	6 bytes	FOR start/current value
\$0006	6 bytes	FOR step value
\$000C	6 bytes	FOR end value
\$0012	2 bytes	number of selections * 4
\$0014	4 bytes	address of just past the END FOR

Integer FOR

\$0000	2 bytes	FOR start/current value
\$0002	2 bytes	FOR step value
\$0004	2 bytes	FOR end value
\$0012	2 bytes	number of selections * 4
\$0014	4 bytes	address of just past the END FOR

LOCal variables

Integer/Float/FOR control variables are stored on the A1 stack as follows

\$00	word	variable reference
\$02	word	type word from name table entry
\$04	long	pointer to value from the name table entry
\$08	word	length of the variable storage area
\$0A	bytes..	storage for the variable integer, 2 bytes float, 6 bytes FOR, 24 bytes

Possible bug in runtimes

```
; Scan the name table we just created
; Loop start
L03D2  move.b  (a0),d2          ;upper byte of type word
      move.w  (a0)+,d1          ;whole type word
      beq.s   L03F6             ;undefined (odd type 00?) converts to
integer

; Normal type
      subq.b  #$03,d2
      beq.s   L03E8             ;sub string, or arrays

      subq.b  #$02,d1
      beq.s   L03F0             ;float

      subq.b  #$01,d1
      beq.s   L03F0             ;integer

; Is this a bug, should it be D2, not D1
      subq.b  #$04,d1
      beq.s   L03FC             ;what's this? converts to float, type 07?

; Anything else or
; sub string, or arrays
L03E8  addq.l  #$06,a0
```

Not happy about this code. Is it wrong, or is it just me?

```
; This code is called from QLIB_OVL  L0014
; Remove an overlay table entry
; A2 is the address of a name table
; D2 is the address of a name table (same as A2)
L0F26  movem.l a3-a4,-(a7)      ;save registers
      movea.l -$0008(a2),a4     ;point at previous name table entry
      movea.l a2,a3             ;A3 is a name table pointer
```

If A3 has been made equal to A2, then won't the next line immediately exit, and the loop will never run?

```
L0F30  cmpa.l  a2,a3
      beq.s   L0F46             ;exit to qlib_ovl

      move.w  (a3),d0           ;get a type word
      cmpi.w  #$012C,d0
      bge.s   L0F4C
```

If D0 had been \$0201 above, then the BGE would have caught it (\$D5) and the next line would be pointless?

```
      cmpi.w  #$0201,d0         ;string variable
      beq.s   L0F60             ;..yes

L0F42  addq.l  #$08,a3           ;point at next name table entry
      bra.s   L0F30             ;loop
```


Compiled jobs A7 stack on startup

With channels and CMD\$

B	Bytes of string	
W	Length of CMD\$	
L	Channel ID	
L	Channel ID	
L	Channel ID	
W	Number of channels	
L	Return address	

<- A7

Negative number of channels
Internally created job for Externals

L	D4	
L	D5	
L	A3	
L	A4	
L	D6	
W	Negative	
L	Return address	

<- A7

The negative number of channels creates a heap entry

L	<- A2
L	
L	
L	-\$C(A2)
L	

Code which does not seem to be ever called in the runtimes

Label

L05F2 - L060C

Possible no longer used code routine 82, which now triggers an 'Internal' error

Just after

L0A42 - L0A52

Just after

L0E4A - L0E56

Just before

L120C

L2312 - L231E

Just before

L1DE6

QLIB_OBJ

Compiler data block. From lv_cdblk in SuperBASIC variables area above.

This area seems to be used for both the reading of the screen pointer, and the reading of the tokenized SuperBASIC program.

compiler data block in QLIB2_asm

gb_chid	\$04	long	channel ID
gb_pos	\$08	long	file position
gb_align	\$0b	byte	bottom byte of file position
gb_line_table	\$0c		line table pointer?

compiler data block in QLIB4_asm

db_tkbuf	\$0c	long	address of tokenized program line buffer, 0 if none
----------	------	------	---

Tokenized program line buffer

tk.head	\$22		length of header
tk_bsize	\$00	long	requested buffer size
tk_fpos	\$04	long	tokenized file pointer
	\$08	long	unused?
tk_lilen	\$0c	word	tokenized line length
tk_hsize	\$0e	long	header size?
tk_rptr	\$12	long	tokenized line running pointer
tk_lstart	\$16	long	tokenized line start
tk_lend	\$1a	long	tokenized line end
tk_fchid	\$1e	long	tokenized file channel ID
tk_hend	\$22		start of tokenized line

;compiler data block in QLIB6_asm

pp_chid	\$00	long	channel ID of window, or -1
			pp_chnr \$02 channel number or -1
pp_sbwnr	\$04	word	sub-window number enclosing the pointer, or -1
pp_xpos	\$06	word	x pixel coordinate of pointer, or -1
pp_ypos	\$08	word	y pixel coordinate of pointer, or -1
pp_kstrk	\$0a	byte	keystroke
pp_kpres	\$0b	byte	key depressed
pp_event	\$0c	long	event vector, all zero except LS byte
qchid	\$10	long	channel ID
qptrrec	\$14	long	WM.RPTR pointer record
qptrwin	\$1a	long	?
	\$2c		
qptrpos	\$34	long	pointer position

Overlays

QLIB_OVL

Overlay table

The overlay table is 128 bytes long. Comprising of 16, 8 byte entries.

Entries 1 to 15 are used by the OVERLAY command. Entry 0 may be something special

layout

\$00	Long	Address of the loaded object file with the externals
\$04	Long	Address of a name table

The **OVERLAY** command creates two new name table types

\$0C00	External Procedure
\$0D00	External Function

possible bug in UNLOAD command

```
; Reached the end of the overlay table
; D6 current job ID
    tst.l    d6                ;are we job 0
    bne.s    lab101ec         ;..no, exit without error

; We are job 0
    movea.l  lv_olbas(a6),a0    ;get address of overlay table

; should this be reclaiming the heap???????????????? MT.RECHP
    moveq    #sms.rrtc,d0       ;MT.RCLCK  Read Real Time Clock
    trap     #1
    clr.l    lv_olbas(a6)       ;set no overlay table anymore

; D1 time in seconds from clock
lab101EC    moveq    #0,d0       ;no error
    rts
```

QLIB_BIN

Just before L08E2,

```
        movea.l d0,a1                ;copy base of system variables
        cmpi.l  #'QLIB',$0024(a1)    QLIB's own spot off PROGD$
        beq.s  L0914

; No program default
L08E2    movea.l a0,a4
```

Is this a clash with a SMSQ/E system variable??

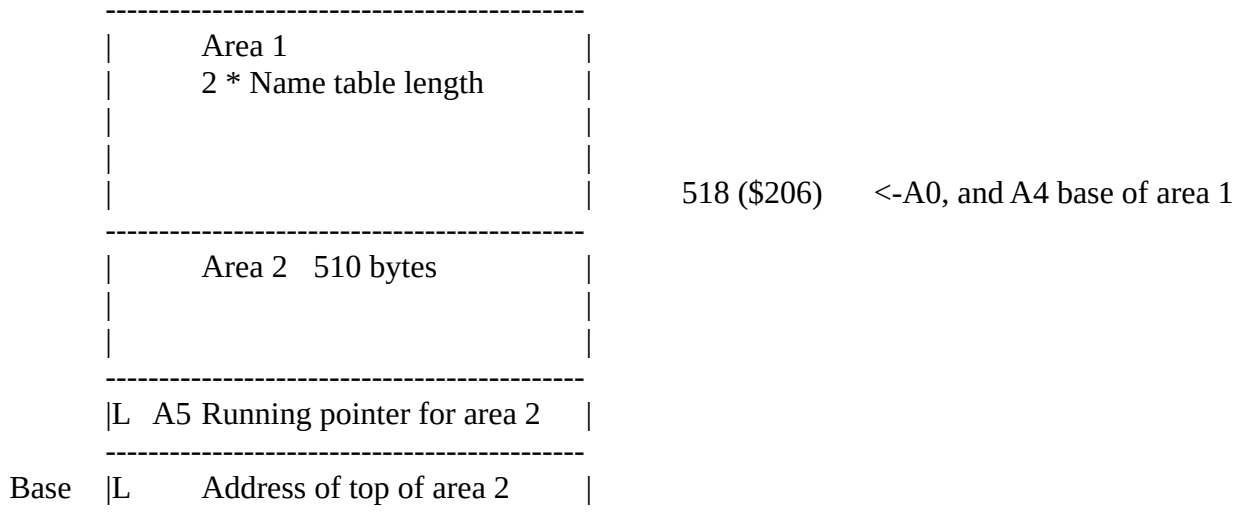
```
; Where does this message get called?
        dc.w    $0013
        dc.b    'Creating work file',$0A,$00

L0892    dc.w    $0012
        dc.b    'Work file created',$0A
```

No label to message

Buffer to be the work file??

The buffer size is (2 * number of name table entries) + 518



QLIB_EXT

Uncalled code

This code does not seem to be called anywhere in QLIB_EXT

*** Part of the LIBERATE command

; This does not seem to be ever called

```
L0806  moveq    #sms.info,d0          ;MT.INF    get system info
      trap     #$01
      move.l    sys_prgd(a0),d1      ;pointer to PRoGram Default
      beq.s     L084C                ;..none, point at SuperBASIC buffer

      moveq     #$38,d0              ;offset to QLIB default load device
      add.l     d0,d1                ;set D1 to it
      bra.s     L0820
```

*** Part of the LIBERATE command

; This does not seem to get called

```
L086A  movem.l  d1-d3/a0-a2,-(a7)
      movea.w   ut.wtext,a2          ;UT.MTXT  Write TEXT
      suba.l    a0,a0                ;channel #0
      jsr       (a2)
      movem.l   (a7)+,d1-d3/a0-a2
      rts
```

Q_ERR

The Q_ERR error table is a linked list of blocks of 512 bytes in the common heap space.

Each block has the following format

	28 entries of 18 bytes	

L	0, or address of next block	

Each 18 byte entry is constructed as follows.

\$00	long word	name table pointer to value, or zero if unused
\$04	movea.l	long word above,a0
	jsr	(a0)
	move.w	d0, lv_qerno(a6)
	moveq	#00,d0
	rts	

QLIB1_BIN

Contains SuperBASIC extensions used by QLIB_OBJ

Adds the following procedures and functions

Procedures

SETUP #ch, data_space

Sets the data space in the file header of the open file on #ch

SETUP #obj, total_data

WINPUT #ch, a, b, c, ...

Read as many words from the channel 'ch', as there are parameters, and place them in the supplied parameters

WINPUT #wrk, NumNames%, area, L_adr

FCOPY #source, #dest

Copies a file from the 'source' channel, to the 'dest' channel

FCOPY #erch, #obj

WWRITE #ch, a, b, c, ...

Writes as many words as there are parameters, to the channel 'ch'

WWRITE #obj, chans_, buffer_, min512, 0, 0, min128

LWRITE #ch, a, b, c, ...

Writes as many long words as there are parameters, to the channel 'ch'

LWRITE #obj, value

RECHP address

Release the allocated 'address' space back into the common heap

RECHP name_table

RNAME addr, var1%, var2%, var3%, var4%

Reads up to 4 words from 'addr' to the supplied parameters

RNAME name_adr(err_num), p0%, struct_base%

WNAME addr,i1%,i2%,i3%,str\$

Append i1%, i2%, i3% and optionally, the characters of str\$ into 'addr' data block

WNAME address, p0%, struct_base%, FOR_ranges%, ExtPFNname\$

TSORT symbols_block

Bubble sort of symbols table

TSORT data_adr

PACK_N v%,n1%,n2%,b1%,b2%,b3%,b4%,b5%,b6%,b7%

Pack the parameters n1% - b7% into v%

PACK_N p0%, p1%, p2%, p3%, p4%, p5%, p6%, p7%, p8%, p9%

UNPACK_N v%,n1%,n2%,b1%,b2%,b3%,b4%,b5%,b6%,b7%

Unpack v% into parameters n1% - b7%

UNPACK_N p0%, p1%, p2%, p3%, p4%, p5%, p6%, p7%, p8%, p9%

Functions

memory = ALCHP(size)

Allocates 'size' bytes from the common heap, and returns the start address of the block

QLIB_area = ALCHP(64)

position = FINDSYMBOL (table_start, symbol, current_position)

Scans the linked list 'table_start', looking for 'symbol'

If found, the offset from the 'current_position' is returned

i = FINDSYMBOL (address, a_sym, RP_ADDR)

result = GENSYMBOL (#ch, table_start, symbol, position)

Scans the linked list 'table_start', looking for 'symbol'

If found, Reads data from 'ch', moves the file pointer and it out again

If not found, Add the 'symbol' to the table, with 'position'

Returns 0, or an error code

i = GENSYMBOL (#obj, address, a_sym, rp_adr)

result = FINDNEXT(address, symbol)

Returns a file position??, or an error code

i = FINDNEXT (address, symbol)

a6 = RA6

Returns the start address of the SuperBASIC variables area

adr236 = PEEK_L ((RA6 + 236))

nt = UNdef(nametable_entry)

Returns true if the name table entry is undefined, otherwise false

IF NOT UNdef(_src\$) : DELETE work\$

mode = RMODE

Returns the current screen mode

IF (RMODE = 8) : MODE 4

Symbol table format

The Symbol table is a linked list of symbol table blocks

Block format

sb_next_block	\$00	long	pointer to next block, zero if the last block
sb_block_end	\$04	long	pointer to the end of the block
sb_last_entry	\$08	long	address of last entry,
sb_entries	\$0C		start of 6 byte entries
	\$00	word	symbol number
	\$02	long	file pointer
sb.entry_size	\$06	entry length (6 bytes)	

QLIB2_BIN

Contains SuperBASIC extensions used by QLIB_OBJ

Adds the following procedures and functions

Procedures

INITgen #ch,pos

Initialize the GENERator block, channel and initial file position

INITgen #obj, 0

GENop opcode

Writes the opcode to the file

GENop 46

GENB a, b, c, ...

Generate an undetermined number of bytes

GENB 3

GENW a, b, c, ...

Generate an undetermined number of words (aligned)

GENW 0, 4

GENL a, b, c, ...

Generate an undetermined number of longs (aligned)

GENL PEEK_L ((ptr + 2))

GENint a, b, c, ...

Generate an undetermined number of ints (not aligned)

GENint sym2%

GENfloat f

Generate 6-byte float

GENfloat sym3

GENsfloat f

Generate 4-byte float

GENsfloat sym3

GENstring s

Generate QDOS style string in the file

GENstring sym4\$

GENtext s

Generate text without size prefix

GENtext name__\$(1 TO 22)

GENntd x

Generate word x * 8

GENntd err_num

SMEM addr, size

Write memory contents from 'addr', of 'size' bytes to the file

```
SMEM b1 , PEEK_W ((b1 - 2))
```

Functions

addr = RP_ADDR

Return current position in stream

```
start_adr = RP_ADDR
```

is_odd = ODD (value)

Return if the given value is odd (1) or not (0)

```
IF ODD (RP_ADDR ) : GENB 0
```

is_short = SHORTF (f)

Check if float can be 32-bit

```
IF SHORTF (sym3) THEN ..
```

GEN block

Uses the QLIB_OBJ compiler data block. from lv_cdblk

gb_chid	\$04	long	channel ID
gb_pos	\$08	long	file position
gb_align	\$0b	byte	bottom byte of file position
gb_line_table	\$0c		line table pointer?

QLIB3_BIN

Contains SuperBASIC extensions used by QLIB_OBJ

Adds the following procedures and functions

Procedures

Q_ERR_ON Q_ERR_ON "wm_rptr"
Q_ERR_OFF Q_ERR_OFF "QLIB_USE"
Q_ERR_LIST

Function

Q_ERR OPEN_IN #wrk, work\$: er = Q_ERR

QLIB3_BIN appears to be the same as QERR_BIN. See the Qliberator manual for descriptions.

QLIB4_BIN

Contains SuperBASIC extensions used by QLIB_OBJ

Adds the following procedures and functions

Procedures

LINT #ch, size, pos
 Create a file buffer of 'size', with a header for the specified channel 'ch'
 Starting from specified file position 'pos'
LINT #wrk, 4096, FPOS (#wrk)

NEXT_TOKEN s1, s2, s3, s4, End_Stmt, End_Line
 Read the next token from the program to be compiled and set the supplied variables
NEXT_TOKEN sym1%, sym2%, sym3, sym4\$, End_of_Stmt%, End_of_Line%

LSET_POS here
 Sets the running pointer in the tokenized program line buffer
LSET_POS here

READ_name #wrk, p0%, LENxPFNname%, ExtPFNname\$
 Reads a name from the channel and sets the last 3 parameters
READ_name #wrk, p0%, LENxPFNname%, ExtPFNname\$

Functions

result = NEXTLINE
 Reads the next tokenized line into a buffer. Returns 1(true) if unable, or on error
IF NEXTLINE

position = LPOS
 Returns the current position in the tokenized program line buffer
here = LPOS

QLIB5_BIN

Contains SuperBASIC extensions used by QLIB_OBJ

Adds the following function

Function

address = b1

Returns the start address of the initialisation code in the final compiled job, or zero. if none
`PEEK_W ((b1 - 2))`

QLIB5_BIN also contains the program initialization code that ends up in the final compiled object file

QLIB6_BIN

Contains SuperBASIC extensions used by QLIB_OBJ

Adds the following procedures and functions

Procedures

WM_Rptr #chan, term, xpos%, ypos%, keystroke%

Read the window manager pointer and fill in the compiler data block

`WM_Rptr #console, 129, x%, y%, Rkey%`

WM_Outln #chan, x-size, y-size, x-origin, y-origin, x-shadow, y-shadow

Set managed outline

`WM_Outln #console, ww, wh, wA, wD, 5, 5`

WM_Sptr #chan, xpos, ypos

Set mouse position relative to window

`WM_Sptr #console, (xx% - 1), (yy% - 1)`

WM_Wrsp #chan, xpos, ypos, sprite-ptr

Draw sprite into window

`WM_wrsp #console, INT((win_dat%(k, 0)/2)), (INT((win_dat%(k, 1)/2))-2), 6`

input_edit\$ #chan, max-len, string

Edit existing string

`input_edit$ #console, L+1, word$`

Dnr #chan, word

Output word value at 0,0

`Dnr #console, line_num`

Functions

win = Fwind (xpos, ypos, window-count, window-int-array (xsize, ysize, xpos, ypos, ?, ?, ?, ?, ?, ?))
k = Fwind (xx%, yy%, panels, win_dat%)

isqueue = Tqueue (#chan)

Is current keyboard queue within specified channel definition block (0/1)

IF (Tqueue (#console) = 0) THEN ...

QLIB6_BIN uses the QLIB_OBJ compiler data block. from lv_cdblk

For reading the pointer position

QLIB data block

pp_chid	\$00	long	channel ID of window, or -1
pp_sbwnr	\$04	word	sub-window number enclosing the pointer, or -1
pp_xpos	\$06	word	x pixel coordinate of pointer, or -1
pp_ypos	\$08	word	y pixel coordinate of pointer, or -1
pp_kstrk	\$0a	byte	keystroke
pp_kpres	\$0b	byte	key depressed
pp_event	\$0c	long	event vector, all zero except LS byte
qchid	\$10	long	channel ID
qptrrec	\$14	long	WM.RPTR pointer record
qptrwin	\$1a	long	?
	\$2c		
qptrpos	\$34	long	pointer position

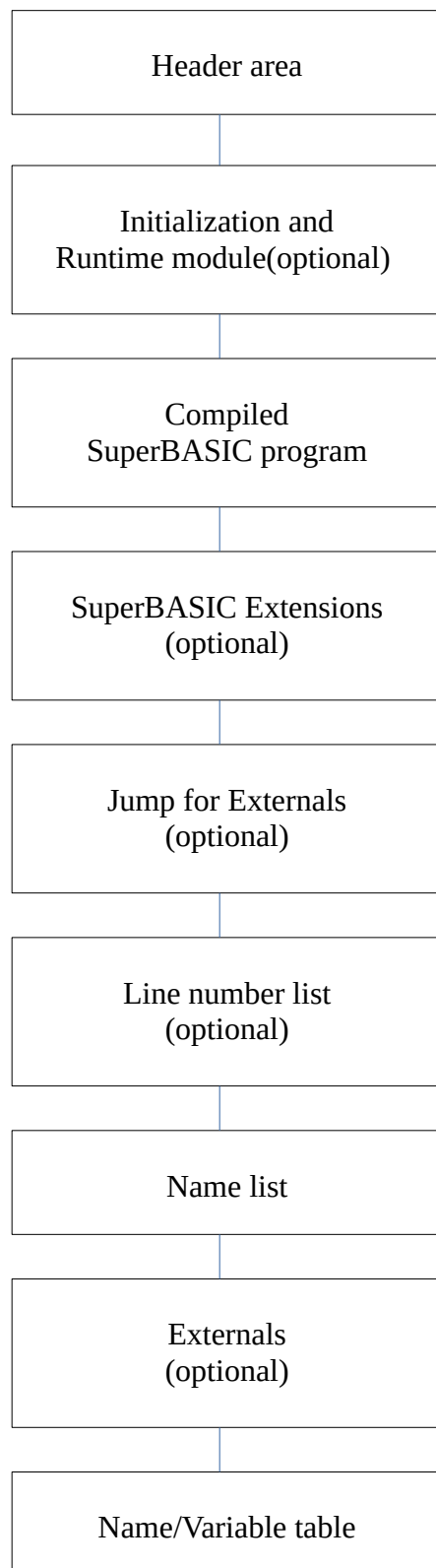
QLIB7_BIN/CONFIG_BIN

Adds the following functions, which return the values set in the config block

Functions

_TICOL	returns the text ink colour
_BPCOL	returns the background paper colour
_SICOL	returns the selected item colour
_DTCOL	returns the deselected item colour
_CUCOL	returns the cursor colour
_NRCHAN	returns the default number of channels
_HEAP	returns the default initial heap size
_STACK	returns the default stack size
_BUFFER	returns the default buffer size
_HELPF	returns the help file name

Layout of compiled object file



Header area

At the start of the executable, just after the job name, there are some offsets into the different parts of the file.

base, is the start of the executable code, and the pointers are offsets from the base address. Offset shown are in decimal

base + 32	Long word pointer to start of encoded SuperBASIC program.
base + 36	Long word pointer to the start of the Name/Variable table.
base + 40	Long word pointer to the start of the list of Line numbers/Offsets.
base + 44	unidentified, Anything less than 4 will open the 3 standard windows 4 is to do with Overlays Anything greater than 4 will cause the WINDS flag to be tested
base + 46	Long word pointer, relative to here, (not the base of the program) to the SuperBASIC DATA start
base + 50	Word, Channel count.
base + 52	Word, Buffer size.
base + 54	Word, Heap chunk size.
base + 56	Long, Heap size.
base + 60	Word, Return stack size.
base + 62	Word, Name table entry of CMD\$ variable. - \$FFFF if none
base + 64	Long word pointer to the start of the Name list.
base + 68	Long word pointer to start of SuperBASIC installed extensions.
base + 72	Long, Data space size. Seems to be invalid in Qlib versions < 3.00
base + 76	Long, Stack size.
base + 80	Byte, Stats. - 0=off, 1=on
base + 81	Byte, AUTOF. - 0=off, 1=on
base + 82	Long word pointer to 'Externals' linked list.
base + 86	unidentified, Seen 2, and 4 here
base + 88	Byte/Word, WINDS - \$FF00=off
base + 90	Long word pointer to Globals

Initialization and Runtime module

This holds the Qlib initialization code, and the runtime code (if included).

Compiled SuperBASIC program

This is the compiled version of the original SuperBASIC program.

In Qlib versions 3.00 and above, the Qliberator version number is stored just before the start of the compiled SuperBASIC as a 4 byte string.

SuperBASIC Extensions (optional)

This holds any SuperBASIC extensions that were included in the original compile.

Jump for Externals

See Externals list below

Line number list

This is a list of the SuperBASIC line numbers (word), and a long word offset from base into the encoded SuperBASIC program.

The last entry starts with \$FFFF, and it is a pointer to the end of the SuperBASIC program.

Beware, some line numbers have the most significant bit set. I have only noticed this in DATA statements.

Name list

This is a list of SuperBASIC keywords used, and may also contain some of the original variable names.

The format of the list is similar to the Name list in QDOS.

byte, length of name, followed by the characters of the name. e.g. \$05 'PRINT'

Note that like in the QL the names may start on odd addresses.

Externals list

If there are any Procedures or Functions marked as 'externals' (REMark \$\$external). They will appear here as a linked list.

The format of the linked list is different for Procedures and Functions, but follows a similar format.

Procedures

Long word pointer to next list item, or 0 if last in list

Standard? SuperBASIC proc/function definition

Word usually \$0002 - number of procedures

Word offset to the NOP (\$4E71) - 2

Bytes Name length (byte), then bytes of real Procedure name padded to a word boundary

Word - end of procedures

Word \$0000 - no functions

Word \$0000 - end of functions

Word \$4E71

Word \$283C (move.l #...,d4)

Long word offset from the start of BASIC, to the start of the Procedure

Functions

Long word pointer to next list item, or 0 if last in list

Standard? SuperBASIC proc/function definition

Word usually \$0002 - no procedures

Word \$0000 - end of procedures

Word \$0000 - no functions

Word offset to the NOP (\$4E71) - 2

Bytes Name length (byte), then bytes of real Procedure name padded to a word boundary

Word \$0000 - end of functions

Word \$4E71

Word \$283C (move.l #...,d4)

Long word offset from the start of BASIC, to the start of the Function

Name/Variable table

This is a list of information on all the names and variables used in the program. The entries in the list are either 1, or 3 bytes long. The 3 byte entries extra word contains an offset from the start of the Name list, to the keyword, or variable name. The table follows the following format, ending in \$FF.

\$00	Undefined	\$10	[word]	Name list undefined
\$01	String variable	\$11	[word]	Name list string variable
\$02	Float variable	\$12	[word]	Name list float variable
\$03	Integer variable	\$13	[word]	Name list integer variable
\$04	String array	\$14	[word]	Name list string array
\$05	Float array	\$15	[word]	Name list float array
\$06	Integer array	\$16	[word]	Name list integer array
\$07	FOR control variable	\$17	[word]	Name list FOR control variable
		\$18	[word]	Name list keyword

Name/Variable references increase in steps of 8. So to find a variable entry in the table, Divide the variable reference by 8, and this will be the entry number in the Name/Variable table.

Entries from \$10 to \$18 refer to names in the name list. The [word] is an offset from the start of the name list.