

## De-Lib Decompiler Technical Notes

This document contains notes for my use on the De-Lib, QLiberator decompiler. But it may be of help for anyone trying to understand how the decompiler program works.

First a few notes on the internal make up of a compiled executable.

The compiled program can be split into four main parts.

The first is a header and the initialization of the compiled program, checking for the existence of the runtime code in memory. Although the runtime code may also be built in here

The second part is the encoded version of the original SuperBASIC program.

The third part, (if present) is a list of the SuperBASIC line numbers, and offsets from the start of the program, to the starts of the individual lines of BASIC in the encoded part.

The fourth part is the Name List, and Name Table area.

This part may also contain externally loaded commands and functions.

Note that unlike Turbo/SuperCharge, codes are byte sized, and not always on word boundaries. So there are some CHR\$(0) used as padding for things that have to be on word boundaries (floats, strings, etc.). But beware, not all CHR\$(0) are padding, some are part of the compiled code.

## Procedures and Functions

While Procedures and Functions with the same names appear in all the decompilers. They may differ in their coding to allow for differences in the decompilers.

### **PROCEDURE SaveMe**

Saves a copy of the program

### **FUNCTION GetString\$(ptr)**

Returns a string of a standard QDOS string (word length then bytes of string) stored in memory at ptr

### **FUNCTION GetInt(ptr)**

Returns a signed integer stored in memory at ptr

### **FUNCTION GetWord(address)**

Returns an unsigned integer stored in memory at address

### **FUNCTION FLT(string\$)**

Converts a 12 character (six byte) hex string into a floating point number

### **FUNCTION FLT\$(n)**

Convert a Floating point number into a 12 character (six byte) hex string  
by PJW & Steven Pool V0.02 03/12/17

### **FUNCTION NameListLen(start)**

Returns the number of the highest name/var list entries  
Start is the address in memory of the keyword table

### **PROCEDURE EmptyPipe (channel)**

Makes sure the pipe is empty. If it is not empty, it is reported to channel, the pipe is emptied displaying the contents of the pipe.

### **FUNCTION GetTOS\$**

Remove and return the item at the rear end of the pipe.

### **PROCEDURE LoadFile (file\$)**

Load the file (file\$) and set base addresses initialBase, base, and filelength.

### **PROCEDURE GetVersion**

Get program version information and set marker variables.

### **PROCEDURE InitMain**

Set up variables for decompiling.

### **PROCEDURE StatementSeperator**

If not a new line output a colon (:) separator

**PROCEDURE CheckSElects (nl)**

Checks for any outstanding SElect processing to deal with. If nl is 1, then we are at the start of a new line

**PROCEDURE CheckELSE (ptr)**

Check to see if an ELSE is needed. ptr is the current program position.  
Return either an empty string or "ELSE"

**FUNCTION CheckENDIF**

Check for pending END IF

**PROCEDURE GetFileNames**

Get the filenames required and set the global variables  
Enter an empty string for the executable to reuse previous values

**FUNCTION FindLineNo (offset)**

Scan the line number list for the offset for the required line number

**PROCEDURE DoProcFun**

Create a DEFINE Proc/Fun line prog points at the offset after a goto

**FUNCTION GetNameVar\$(ident)**

Returns a name, or a variable. If actual variable name does not exist in namelist  
then return in the form, varXXXX

**FUNCTION LineEmpty(num)**

Checks to see if the supplied line number is empty. Returns 1 if it is, or 0 if not or does not exist

**FUNCTION GetFORvalue\$**

Read the next FOR value and return as a string

**PROCEDURE Listnames**

List all the names and variables in the currently loaded compiled program to a file  
(De compiling of the program must have been started at least once)

**PROCEDURE CheckExtensions**

Check to see if there are any embedded extensions. If so offer to extract them

**PROCEDURE ListTable (ch,table)**

List the Procedure and Function names in the table, to the channel ch

**PROCEDURE ListLineNo**

List all the line numbers (if they exist) in the currently loaded compiled program to a file

**PROCEDURE CheckExternals**

Check to see if there are any Procedures or Functions assigned as 'Externals', and list them in the log file

## Global variables

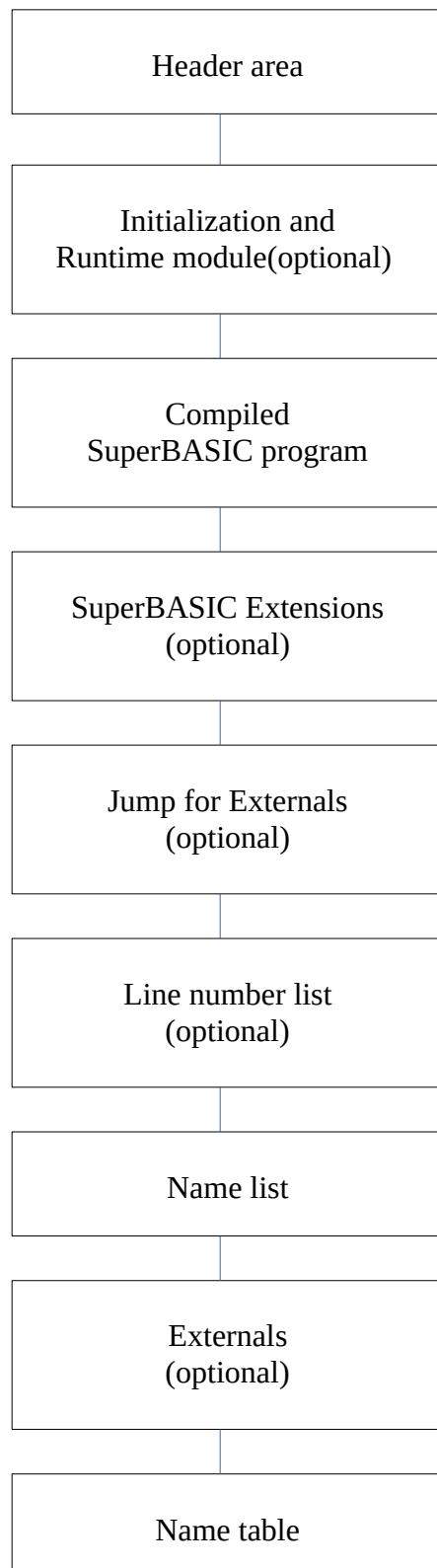
filename\$	Name of the executable file
filelength	Length of executable file.
fileData	The executables data space.
initalBase	Loaded base address, may be different to 'base' if Qemulator header found.
base	Start of the executable.
finish	End of the executable.
progOffset	Offset from start of executable to start of BASIC program area.
prog	Running pointer for the BASIC program.
basicProgStart	Start of BASIC program area.
varInitOffset	Offset from start of executable to start of variable init area.
varInitStart	Start of variable init area.
keywords	Start of keyword table.
progEnd	End of BASIC program area.
jobName\$	Executables job name.
lineNoExist	Flag for the existence of a line number table. 0=No line number table 1=Line number table exists
externalProcFunlist	Greater than 0, if 'External' Procedures and Functions are included.
lineNumberList	Address of the start of the line number table. If it exists.
lineNumberListEnd	Address of the end of the line number list. If it exists.
keywordOffsetAddress	that is a base for accessing Keywords. Not a pointer to the keyword list itself.
lineCount	Number of program lines decompiled.
lineNo	Line number of line currently de compiling.
key	Holds the current operation prefix code.
ch	Channel to send decompiled output to.
logCh	Channel to send Errors & log to.
parmCount	Used in keyword entries for number of parameters on the stack
cmdType	Used in keyword entries for type of call 1 = Procedure
procFun\$	The name of the current Procedure or Function
pcount	Number of items in the pipe.
InIF	Number of levels of nested IF's.
inFun	Number of items of nested function calls
newLine	0 = Not at the start of a new line.
InSel	Number of levels of nested SElect's.

selString\$	Current SElect selection string.
selFlag	Set when doing a SElection selection string.
selTOFlag	Flag that indicates that a 'TO' has just been done on a SElect ON
endWhenAdd	Address of END WHEN when in a WHEN ERRor
ReadFlag	Flag that is set during a READ into an array
forVar\$	Control variable for a FOR (may not be used)
forFlag	Flag that is set when a FOR selection is being set up. It disables any GO TO's during the selection setting up.
forCount	Stores the stack position during FOR selection.
forSelCount	Number of items in a FOR selection.
forSelString\$	Used to hold a FOR selection e.g. 1,3,5 TO 10

## Arrays

nameList()	Array of offsets for keywords at end of code.
selStack(10,2)	<p>Holds information on nested SElect's</p> <p>Second index -        0 Pointer to address of END SElect</p> <p>                         1 Pointer to start of code block</p> <p>                         2 Select variable</p>
funLevel(x,1)	<p>Array holding information on namelist keyword functions, or Proc/Fun calls.</p> <p>Second index -        0 FunFlag        0 for a command/procedure</p> <p>   1 for a function</p> <p>                         1 FunParmCount</p>
ifStack(x,1)	<p>Array holding information on nested IF..THEN's</p> <p>Second index -        0 Destination address of ELSE or END IF</p> <p>                         1 A flag for the ELSE status</p> <p>                             0 - Inhibit ELSE check on next GO TO (\$CA) code</p> <p>                             1 - Destination address. May be ELSE of END IF</p>

## Layout of compiled object file



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.

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, should be in range 2->4
base + 46	Long word pointer, relative to here, (not the base of the program) to the SuperBASIC DATA
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.
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 .
base + 76	Long, Stack size.
base + 80	Byte, Stats.
base + 81	Byte, AUTOF.
base + 82	Long word pointer to 'Externals' linked list.
base + 86	
base + 88	Byte/Word, WINDS

## 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 the original variable names.

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

## 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
Word	usually \$0002
Word	if not zero, then this is an offset to the NOP (\$4E71) - 2
Bytes	Name length (byte), then bytes of real Procedure name padded to a word boundary
Long word	
Word	
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
Word	usually \$0002
Word	0
Long word	
Bytes	Name length (byte), then bytes of real Procedure name padded to a word boundary
Word	
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.

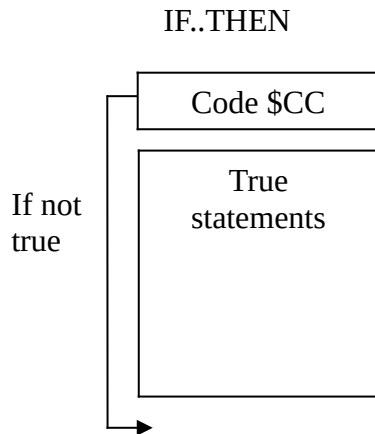
The table starts with a \$03, and ends with a \$FF (or maybe a zero? V3.35)

\$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

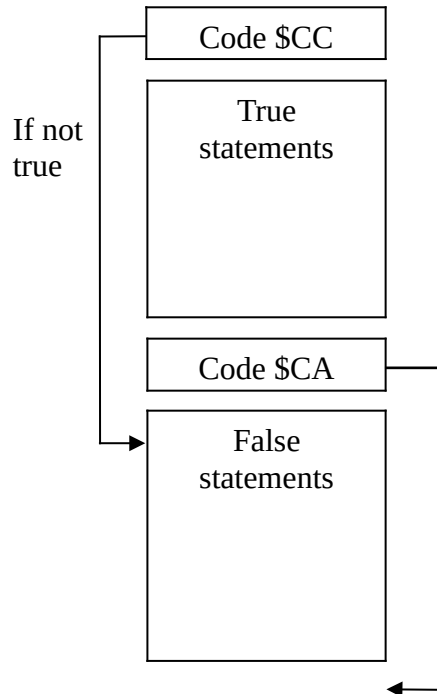
Name/Variable references start at 8, and increase in steps of 8. So to find a variable entry in the table, Divide the variable reference by 8, and this will be the entry number in the Name/Variable table.

## Format of program storage in the compiled program

### IF ..THEN



### IF..THEN..ELSE



### REPEAT loops

On the REPEAT line, the control variable is set to 0(zero), but does not seem to be referenced to again in the loop.

The END REPEAT is a GO TO back to just after the assignment of the control variable.

NEXT is as above, and EXIT is a GO TO to just past the END REPEAT.

## FOR .. END FOR

FOR z = x TO y

\$9E	[word]	Control variable, Word is variable reference
\$8A	[word]	Start, Word is integer? value
\$8A	[word]	End, Word is integer? value
\$8A	[word]	Step, Word is integer? value
\$A4	[word] [long] [long]	Word is variable reference, first Long is a pointer to the start of the statements, second Long is a pointer to just past the END FOR

END FOR

\$A2	[word] [long] [long]	Word is variable reference, first Long is a pointer to the start of the statements, second Long is a pointer to just past the END FOR
------	----------------------	---

More complex FOR's need looking into. Like FOR x = 1,3,5 TO 10,12,  
and FOR x= a TO 13.7 STEP 0.3

## SElect

SElect is a bit of a problem area. A simple ON x=3 is OK. But a ON x=1 TO 5, is awkward, as the 1 TO 5 is converted into what looks like an IF..ELSE..THEN to the decompiler.

### A simple SElect ON

**SElect ON y**

\$CA	[long]	Jump to the first test. (skip over the next \$CA)
------	--------	---

**ON y = 10**

\$CA	[long]	Jump to the END SElect
------	--------	------------------------

Do the y=10 test

\$D0	[long]	If the test is true, jump to the start of Statements
------	--------	--

\$CA	[long]	Otherwise, jump over the statements to Next test, or END SElect
------	--------	---

Statements

\$CA	[long]	Jump to the END SElect
------	--------	------------------------

Next test

**ON y = 84,116**

\$CA	[long]	Jump to the END SElect
------	--------	------------------------

Do the y=84 test

\$D0	[long]	If the test is true, jump to the start of Statements
------	--------	--

Do the y=116 test

\$D0	[long]	If the test is true, jump to the start of Statements
------	--------	--

\$CA	[long]	Otherwise, jump over the statements to Next test, or END SElect
------	--------	---

Statements

\$CA	[long]	Jump to the END SElect
------	--------	------------------------

Next test

## A SElect ON with a TO

### SElect ON y

\$CA [long] Jump to the first test. (skip over the next \$CA)

### ON y = 1 TO 10

Do a test that y >= 1

\$CC [long] If not true jump to the GO TO (\$CA)

Do a test that y > 10

\$CC [long] If not true jump over the next GO TO (\$CA)

\$CA [long] Jump over the statements to the next test, or END SElect  
Statements

## Procedure and Function definitions

\$CA [long] A GO TO to just past the END DEFine  
\$76 [byte] [words] The byte is the number of parameters, and the Words are the parameters

\$60 RETURN a vaule on the stack (FuNctions)  
\$5E RETURN/END DEFine Procedure  
\$5C \$02 END DEFine Function, I don't know what the \$02 means

## Local variables

\$98 [word] Float, Word is variable reference  
9C [word] String, Word is variable reference

## DATA statements

A DATA line starts with a \$CA code pointing to just past the end of the DATA statements

The DATA value is placed on the stack, followed by \$C0 code. This is then repeated for however many more DATA values there are.

Finally there is another \$CA code pointing to either the next DATA line?, or pointing to the end of the program.