Sinclair BASIC tokenized file

From Just Solve the File Format Problem

File Format	
Name	Sinclair BASIC tokenized file
Ontology	▪ Electronic File Formats ▪ Source code ▪ Tokenized BASIC ▪ **Sinclair BASIC tokenized file**

Sinclair BASIC is a dialect of the BASIC programming language created by Nine Tiles Networks Ltd and used in the 8-bit home computers from Sinclair Research and Timex Sinclair.

The original 4KB version was developed for the Sinclair ZX80, followed by an 8KB version for the ZX81 and 16KB version for ZX Spectrum.

Some unusual features of the Sinclair BASIC:

- There were keys on the keyboard for each BASIC keyword. For example, pressing P caused the entire command PRINT to appear. Some commands needed multiple keypresses to enter, For example, BEEP was keyed by pressing CAPS SHIFT plus SYMBOL SHIFT to access extended mode, keeping SYMBOL SHIFT held down and pressing Z.
- When programs were SAVEd, the file written to disk or tape contained all of BASIC's internal state information, including the values of any defined basic variables, as well as the BASIC tokens.

Contents

Format details

BASIC File Layout

On a ZX81, a saved BASIC file is a snapshot of the computer memory from memory location 16393 through to the end of the variable table. There is no header.

Address	Name	Description
16393	VERSN	0 Identifies ZX81 BASIC in saved programs.
16394	E_PPC	Number of current line (with program cursor).
16396	D_FILE	Pointer to the start of the 'Display file', i.e. what is being displayed on screen
16398	DF_CC	Address of PRINT position in display file. Can be poked so that PRINT output is sent elsewhere.
16400	VARS	Pointer to start of BASIC Variable table
16402	DEST	Address of variable in assignment.
16404	E_LINE	Pointer to line currently being entered
16406	CH_ADD	Address of the next character to be interpreted: the character after the argument of PEEK, or the NEWLINE at the end of a POKE statement.
16408	X_PTR	Address of the character preceding the marker.
16410	STKBOT	pointer to start (bottom) of stack
16412	STKEND	pointer to end (top) of stack
16414	BERG	Calculator's b register.
16415	MEM	Address of area used for calculator's memory. (Usually MEMBOT, but not always.)
16417		not used
16418	DF_SZ	The number of lines (including one blank line) in the lower part of the screen.
16419	S_TOP	The number of the top program line in automatic listings.
16421	LAST_K	Shows which keys pressed.
16423		Debounce status of keyboard.
16424	MARGIN	Number of blank lines above or below picture: 55 in Britain, 31 in America.
16425	NXTLIN	Address of next program line to be executed.
16427	OLDPPC	Line number of which CONT jumps.
16429	FLAGX	Various flags.
16430	STRLEN	Length of string type destination in assignment.
16432	T_ADDR	Address of next item in syntax table (very unlikely to be useful).
16434	SEED	The seed for RND. This is the variable that is set by RAND.
16436	FRAMES	Counts the frames displayed on the television. Bit 15 is 1. Bits 0 to 14 are decremented for each frame set to the television. This can be used for timing, but PAUSE also uses it. PAUSE resets to 0 bit 15, & puts in bits 0 to 14 the length of the pause. When these have been counted down to zero, the pause stops. If the pause stops because of a key depression, bit 15 is set to 1 again.
16438	COORDS	x-coordinate of last point PLOTted.
16439		y-coordinate of last point PLOTted.
16440	PR_CC	Less significant byte of address of next position for LPRINT to print as (in PRBUFF).
16441	S_POSN	Column number for PRINT position.
16442		Line number for PRINT position.
16443	CDFLAG	Various flags. Bit 7 is on (1) during compute & display mode.
16444	PRBUFF	Printer buffer (33rd character is NEWLINE).
16477	MEMBOT	Calculator's memory area; used to store numbers that cannot conveniently be put on the calculator stack.
16507		not used
16509		First BASIC line.

BASIC lines

Each BASIC line is stored as:

- 2 byte line number (in big-endian format)
- 2 byte length of text including NEWLINE (in little endian format, length "excludes" the line number and length, i.e. to skip between lines you add "length of text" +4 bytes.
- text (BASIC tokens)
- NEWLINE (0x76 on ZX80/81, 0x0D on Spectrum)

When a numeric constant is included in the text of a BASIC line, an ASCII string displaying the constant value will be inserted, followed by one of two tokens (indicating floating-point or integral) and then the number in 5-byte numeric format.

BASIC Variables Table

Following the last BASIC line comes the variables table. Each entry in this table is of varying length and format.

The first byte in each entry is the variable name, of which the upper 3 bits indicate the variable type.

Most types of variables can only have a one-character name: A to Z for numerics, A$ to Z$ for strings. Numeric variables can have a multi-character names beginning with A-Z and continuing with A-Z or 0-9, e.g. `F00` or `BAR`. Names are case-insensitive and whitespace insensitive (`hello world` is the same variable as `HeLlOwOrL d`). Numeric variables and FOR-NEXT control variables share the same namespace, but no other types do. Numeric variable A, string variable A$, numeric array A(10) and string array A$(10) can all coexist under the name "A"[1]

First byte	Format	Examples
011nnnnn	numeric variable • 1 byte: variable name (0x61 to 0x7A) • 5 bytes: numeric value	• `61 00 00 01 00 00` → `LET a=1` • `69 00 FF 02 00 00` → `LET i=-65534` • `6A 00 00 FE FF 00` → `LET j=65534` • `62 93 3C 61 4E 00` → `LET b=12345678` ◦ 12345678 = 0xBC614E • `63 93 BC 61 4E 00` → `LET c=-12345678` • `65 82 2D F8 4C AD` → `LET e=2.71828`

101nnnnn	numeric variable with multi-character name
	• 1 byte: first character of variable name (0xA1 to 0xBA)
	• n bytes: remainder of name (A-Z, a-z, 0-9 allowed, final character has high bit set)
	• 5 bytes: numeric value
		• `B0 69 E5 82 49 4A F0 41` → `LET pie=3.1452`
		  ◦ "p"=0x10 ORed with 0xA0
		  ◦ "i" is regular ASCII (0x69)
		  ◦ "e" is regular ASCII (0x65) ORed with 0x80 to indicate the end of the variable name

010nnnnn	string variable • 1 byte: variable name (0x41 to 0x5A) • 2 bytes: little-endian string length (*n*) • *n* bytes: string value (the character set is mostly ASCII)[2]	• `41 04 00 54 45 53 54` → `LET a$="TEST"`
100nnnnn	numeric array • 1 byte: array name (0x81 to 0x9A) • 2 bytes: little-endian size of data to follow in bytes, so you can easily skip to the next variable without computing full array size • 1 byte: number of dimensions in the array (1-255?) • 2 bytes: little-endian valid range of first dimension e.g. `0A 00` is valid range 1-10 • ... 2 bytes more for every other dimension to declare the rest of the dimension ranges • ... array values (each a 5-byte number) in C-style order, iterating rightmost index first and the leftmost index last, e.g. `DIM a(2,3,4)` will store values in this order: a(1,1,1), a(1,1,2), a(1,1,3), (1,1,4), a(1,2,1), a(1,2,2), a(1,2,3), a(1,2,4), a(1,3,1), a(1,3,2), a(1,3,3), a(1,3,4), a(2,1,1), ...	• `81 1C 00 01 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00` → `DIM a(5)` • `81 19 00 02 02 00 02 00 00 00 01 00 00 00 00 02 00 00 00 00 03 00 00 00 00 04 00 00` → `DIM a(2,2); LET a(1,1)=1; LET a(1,2)=2; LET a(2,1)=3; LET a(2,2)=4`
110nnnnn	character array • 1 byte: array name (0xC1 to 0xDA) • 2 bytes: little-endian size of data to follow in bytes, so you can easily skip to the next variable without computing full array size • 1 byte: number of dimensions in the array (1-255?) • 2 bytes: little-endian valid range of first dimension e.g. `0A 00` is valid range 1-10 • ... 2 bytes more for every other dimension to declare the rest of the dimension ranges • ... array values (each a 1-byte character) in C-style order, iterating rightmost index first and the leftmost index last A single-dimensional character array acts like a string, but has a fixed length. If it is set to a shorter string, the remaining space in the array will be padded with spaces (0x20). In general, an *n* dimensional character array where n>1 acts like an *n*-1 dimensional string array, e.g. `DIM a$(5,10); LET a$(1)="FOO"; LET a$(2)="BAR"` is valid, and sets a$(1,1) to a$(1,10) to "FOO ", and sets a$(2,1) to a$(2,10) to "BAR "	• `C8 12 00 01 0F 00 68 65 6C 6C 6F 2C 20 77 6F 72 6C 64 20 20 20` → `DIM h$(15); LET h$="hello, world"` • `D7 0F 00 02 02 00 05 00 68 65 6C 6C 6F 77 6F 72 6C 64` → `DIM w$(2,5); LET w$(1)="hello"; LET w$(2)="world"`
111nnnnnn	control variable of a FOR-NEXT loop. • 1 byte: variable name (0xE1 to 0xFA)	• `E9 00 00 01 00 00 00 00 05 00 00 00 00 01 00 00 FF FE 02` → `FOR i=1`

	▪ 5 bytes: current value (number) ▪ 5 bytes: loop end limit (number) ▪ 5 bytes: loop step increment (number) ▪ 2 bytes: little-endian line number where the FOR loop was started ▪ 1 byte: statement number within line where the FOR loop was started	`TO 5` (before loop has run) ▪ `E9 00 00 06 00 00 00 00 05 00 00 00 00 01 00 00 FF FE 02` → `FOR i=1 TO 5` (after loop has run) ▪ `E9 00 00 01 00 00 00 00 0A 00 00 00 00 02 00 00 FF FE 02` → `FOR i=1 TO 10 STEP 2` ▪ `E9 81 0C CC CC CD 83 76 66 66 66 7D 4C CC CC CC FE FF 02` → `FOR i=1.1 TO 7.7 STEP .1`

5-byte numeric format

Numbers have one of two formats:[3] integers between -65535 and +65535 (inclusive) are in an "integral" format, while all other numbers are in "floating point" format. The value 0 can't be represented by the floating point format, so is always stored as an integral. The token 0x7E before the number indicates floating-point, and 0x0E indicates an integral number.

With "integral" format:

- 1 byte: always 0
- 1 byte: 0 if the number is positive or -1 (0xFF) if the number is negative
- 2 bytes: little-endian unsigned integer from 0 to 65535. Subtract 65536 from this if number is flagged as negative
- 1 byte: always 0

With "floating point" format:

- 1 byte: exponent + 128 (0 → e=-128, 255 → e=127)
- 4 bytes: big-endian mantissa

The number has to be normalised so that its most significant mantissa bit is *always* 1. This bit is then assumed to be 1 and is overwritten with a sign bit: cleared to 0 for positive numbers and set to 1 for negative numbers

Floating point examples

Representation	Value
`7F 7F FF FF FF`	0.5
`7E 7F FF FF FF`	0.25
`7D 7F FF FF FF`	0.125
`91 00 00 00 00`	65536
`A0 07 65 43 21`	2271560481 (0x87654321)
`7F FF FF FF FF`	-0.5
`7E FF FF FF FF`	-0.25
`7D FF FF FF FF`	-0.125
`91 80 00 00 00`	-65536
`A0 87 65 43 21`	-2271560481 (-0x87654321)

Integral examples

Representation	Value
`00 FF 01 00 00`	-65535
`00 FF 02 00 00`	-65534
`00 FF FE FF 00`	-2
`00 FF FF FF 00`	-1
`00 00 00 00 00`	0
`00 00 01 00 00`	1
`00 00 02 00 00`	2
`00 00 FE FF 00`	65534
`00 00 FF FF 00`	65535

ZX81 BASIC Tokens

Token (Decimal)	Description
0	
11	"
12	£
13	$
14	:
15	?
16	(
17	)
18	>
19	<
20	=
21	+
22	-
23	*
24	/
25	;
26	,
27	.
28	0
29	1
30	2
31	3
32	4
33	5
34	6
35	7
36	8
37	9
38	A
39	B
40	C
41	D
42	E
43	F
44	G
45	H
46	I
47	J
48	K
49	L
50	M
51	N
52	O
53	P
54	Q
55	R
56	S
57	T

58	U
59	V
60	W
61	X
62	Y
63	Z
64	RND
65	INKEY$
66	PI
112	<cursor up>
113	<cursor down>
114	<cursor left>
115	<cursor right>
116	GRAPHICS
117	EDIT
118	NEWLINE
119	RUBOUT
120	/
121	FUNCTION
127	cursor
128	
139	"
140	£
141	$
142	:
143	?
144	(
145	)
146	>
147	<
148	=
149	+
150	-
151	*
152	/
153	;
154	-
155	.
156	0
157	1
158	2
159	3
160	4
161	5
162	6
163	7
164	8
165	9
166	A

167	B
168	C
169	D
170	E
171	F
172	G
173	H
174	I
175	J
176	K
177	L
178	M
179	N
180	O
181	P
182	Q
183	R
184	S
185	T
186	U
187	V
188	W
189	X
190	Y
191	Z
192	""
193	AT
194	TAB
195	?
196	CODE
197	VAL
198	LEN
199	SIN
200	COS
201	TAN
202	ASN
203	ACS
204	ATN
205	LN
206	EXP
207	INT
208	SQR
209	SGN
210	ABS
211	PEEK
212	USR
213	STR$
214	CHR$
215	NOT

216	**
217	OR
218	AND
219	<=
220	>=
221	<>
222	THEN
223	TO
224	STEP
225	LPRINT
226	LLIST
227	STOP
228	SLOW
229	FAST
230	NEW
231	SCROLL
232	CONT
233	DIM
234	REM
235	FOR
236	GOTO
237	GOSUB
238	INPUT
239	LOAD
240	LIST
241	LET
242	PAUSE
243	NEXT
244	POKE
245	PRINT
246	PLOT
247	RUN
248	SAVE
249	RAND
250	IF
251	CLS
252	UNPLOT
253	CLEAR
254	RETURN
255	COPY

ZX Spectrum 48/128 BASIC Tokens

Token (Hex)	Description
0xA3	SPECTRUM[1]
0xA4	PLAY[1]
0xA5	RND
0xA6	INKEY$
0xA7	PI
0xA8	FN
0xA9	POINT
0xAA	SCREEN$
0xAB	ATTR
0xAC	AT
0xAD	TAB
0xAE	VAL$
0xAF	CODE
0xB0	VAL
0xB1	LEN
0xB2	SIN
0xB3	COS
0xB4	TAN
0xB5	ASN
0xB6	ACS
0xB7	ATN
0xB8	LN
0xB9	EXP
0xBA	INT
0xBB	SQR
0xBC	SGN
0xBD	ABS
0xBE	PEEK
0xBF	IN
0xC0	USR
0xC1	STR$
0xC2	CHR$
0xC3	NOT
0xC4	BIN
0xC5	OR
0xC6	AND
0xC7	<=
0xC8	>=
0xC9	<>
0xCA	LINE
0xCB	THEN
0xCC	TO
0xCD	STEP
0xCE	DEF FN
0xCF	CAT
0xD0	FORMAT
0xD1	MOVE
0xD2	ERASE

0xD3	OPEN #
0xD4	CLOSE #
0xD5	MERGE
0xD6	VERIFY
0xD7	BEEP
0xD8	CIRCLE
0xD9	INK
0xDA	PAPER
0xDB	FLASH
0xDC	BRIGHT
0xDD	INVERSE
0xDE	OVER
0xDF	OUT
0xE0	LPRINT
0xE1	LLIST
0xE2	STOP
0xE3	READ
0xE4	DATA
0xE5	RESTORE
0xE6	NEW
0xE7	BORDER
0xE8	CONTINUE
0xE9	DIM
0xEA	REM
0xEB	FOR
0xEC	GO TO
0xED	GO SUB
0xEE	INPUT
0xEF	LOAD
0xF0	LIST
0xF1	LET
0xF2	PAUSE
0xF3	NEXT
0xF4	POKE
0xF5	PRINT
0xF6	PLOT
0xF7	RUN
0xF8	SAVE
0xF9	RANDOMIZE
0xFA	IF
0xFB	CLS
0xFC	DRAW
0xFD	CLEAR
0xFE	RETURN
0xFF	COPY

[1]: This token is only available in 128 BASIC.

Links

- ZX81 BASIC Programming (http://www.worldofspectrum.org/ZX81BasicProgramming/)
- Sinclair ZX81 review (2012) (http://www.trustedreviews.com/sinclair-zx81_Desktop-PC_review)
- Sinclair ZX81 BASIC Programming Manual (http://www.old-computers.com/museum/docs.asp?c=263)

References

1. ↑ http://www.worldofspectrum.org/ZXBasicManual/zxmanchap7.html
2. ↑ http://www.worldofspectrum.org/ZXBasicManual/zxmanappa.html
3. ↑ http://www.worldofspectrum.org/ZXBasicManual/zxmanchap24.html

Categories: File Formats | Electronic File Formats | Source code | Tokenized BASIC | ZX Spectrum

---