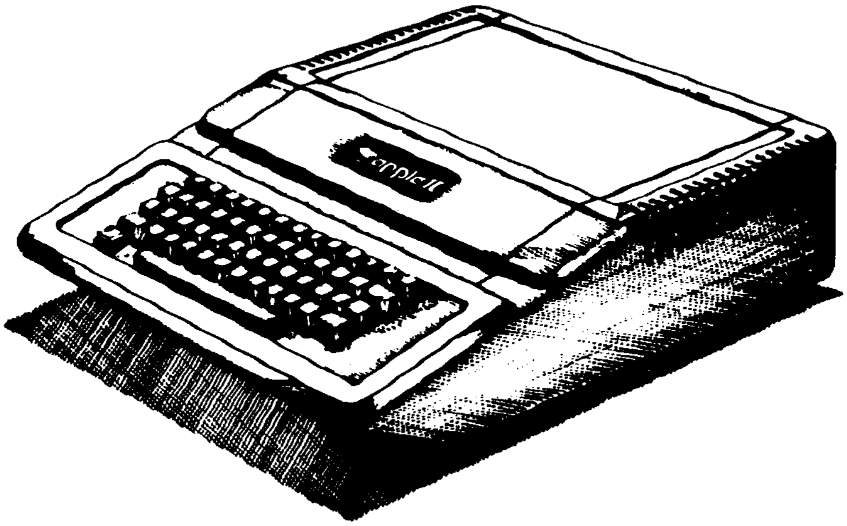
Apple 2 Computer Technical Information



Apple][Computer Family Information

AppleSoft BASIC Info:

Apple Soft Internal Structure

Mesztenyi – Call APPLE – Jan 1982

Document # 45

Ex Libris David T. Craig

"DTCA2DOC-045-00.PICT" 141 KB 2001-03-29 dpi: 300h x 300v pix: 2037h x 2790v

FROM THE VERY CORE OF APPLE:

Applesoft Internal Structure

By C.K. Mesztenyi/Washington Apple Pi

INTRODUCTION

THIS article attempts to describe the overall structure of Applesoft in the ROM space $D000-F7FF; it may be considered as a preceding chapter to "Applesoft Internals," hereinafter referred to as "Crossley", and gives descriptions of many subroutines and zero page usage. Crossley and other abbreviated references, known herein by Lingwood, Mestztenyi and Golding, may be found at the conclusion of this article.

Before going into details, I must define certain terms for the sake of this article which may be very confusing in the Applesoft Manual. These terms are the "statement," "command," "instruction," "line number" and "line." The first three of these are used somewhat interchangeably in the Manual. It refers to REM and Assignment or LET statements in Chapter 1, lists them as Commands together with ABS in Appendix O, and assumes them to be instructions in Chapter 2 and Appendix N. I do not intend to clear all these confusions and errors in the syntactic definition and subsequently used terminology. instead, the following syntactic definitions will be used here with the hope that I will not confuse the issue further. These definitions are as follows:

statement := end-st / for-st/ / ... / new-st
let-st := assign-st / LET assign-st
compound-statement:= statement [CR] statement :
labeled-statement := linenumber compound-statement

For example, I define a "statement" as any of the 64 statements with the keyword "end," "for,"... as listed in the keyword column of the Statement Type Entry Table; the syntactic rules of these individual statements are given in the Manual under their descriptions. The compound-statement is a list of [simple] statements separated by a ":", while the labeled-statement is a line number followed by the compound-statement which the Manual defined as "line." [CR] stands for carriage return.

With these definitions, one can state that a compound-statement is a program in immediate mode, while a labeled-statement is a program part in deferred mode.

1. DATA STRUCTURE

The data areas used by Applesoft reside:

1. Flags and temporaries on Zero page.
2. Five Tables in memory $D000-D364.
3. Scattered (locally used) data interspersed in the program area $D365-F7FF.
4. Zero page load data in memory $F10B-F126.
5. Stored program normally from memory address $0801.
6. Variable areas.

1.1 Zero Page

The zero page use is described in (Applesoft, Basic Programming Reference Manual pp.140-141). Further information may be found in [Crossley], [Mesztenyi], and [Lingwood].

1.2 Tables

The five tables residing in $D000-D364 are as follows:

$D000-D07F = Statement Type Entry Table.
$D080-D0B1 = Function Entry Table.
$D0B2- D0CF=Operator Tag and Entry Table.
$D0D0- D25F=Keyword Token Table.
$D260-D364 = ASCII Messages.

APPLESOFT INTERNAL STRUCTURE

The Statement Type Entry Table is used to recognize statements and to obtain the proper entry points in the program area. It consists of 64 two byte entries containing the entry point low-high addresses minus one. The order of the 64 entries correspond to the tokens, 128 to 191, assigned to the keywords END to NEW, as given in (Applesoft, Basic Programming Reference Manual, p.121). Table 1 summarizes these data, giving the actual entry point addresses.

TABLE 1
Statement Type Entry Table From $D00-D07F

Hex Token	Key Word	Entry Point	Hex Token	Key Word	Entry Point
$80	END	$D870	$A0	COLOR=	$F24F
$81	FOR	$D766	$A1	POP	$D96B
$82	NEXT	$DCF9	$A2	VTAB	$F256
$83	DATA	$D995	$A3	HIMEM:	$F286
$84	INPUT	$DBB2	$A4	LOMEM:	$F2A6
$85	DEL	$F331	$A5	ONERR	$F2CB
$86	DIM	$DFD9	$A6	RESUME	$F318
$87	READ	$DBE2	$A7	RECALL	$F3BC
$88	GR	$F390	$A8	STORE	$F39F
$89	TEXT	$F399	$A9	SPEED=	$F262
$8A	PR#	$F1E5	$AA	LET	$DA46
$8B	IN#	$F1DE	$AB	GOTO	$D93E
$8C	CALL	$F1D5	$AC	RUN	$D912
$8D	PLOT	$F225	$AD	IF	$D9C9
$8E	HLIN	$F232	$AE	RESTORE	$D849
$8F	VLIN	$F241	$AF	&	$03F5
$90	HGR2	$F3D8	$B0	GOUB	$D921
$91	HGR	$F3E2	$B1	RETURN	$D96B
$92	HCOLOR=	$F6E9	$B2	REM	$D9DC
$93	HPLOT	$F6FE	$B3	STOP	$D86E
$94	DRAW	$F769	$B4	ON	$D9EC
$95	XDRAW	$F76F	$B5	WAIT	$E784
$96	HTAB	$F7E7	$B6	LOAD	$D8C9
$97	HOME	$FC58	$B7	SAVE	$D8B0
$98	ROT=	$F721	$B8	DEF	$E313
$99	SCALE=	$F727	$B9	POKE	$E77B
$9A	SHLOAD	$F775	$BA	PRINT	$DAD5
$9B	TRACE	$F26D	$BB	CONT	$D896
$9C	NOTRACE	$F26F	$BC	LIST	$D6A5
$9D	NORMAL	$F273	$BD	CLEAR	$D66A
$9E	INVERSE	$F277	$BE	GET	$DBA0
$9F	FLASH	$F280	$BF	NEW	$D649

TABLE 2
B
Function Entry Table From $D080-D081

Hex Token	Key Word	Entry Point	Hex Token	Key Word	Entry Point
$D2	SGN	$EB90	$DF	SIN	$EFF1
$D3	INT	$EC23	$E0	TAN	$F03A
$D4	ABS	$EBAF	$E1	ATN	$F09E
$D5	USR	$000A	$E2	PEEK	$E764
$D6	FRE	$E2DE	$E3	LEN	$E6D6
$D7	SCRN(	$D412	$E4	STR$	$E3C5
$D8	PDL	$DFCD	$E5	VAL	$E707
$D9	POS	$E2FF	$E6	ASC	$E6E5
$DA	SQR	$EE8D	$E7	CHR$	$E646
$DB	RND	$EFAE	$E8	LEFT$	$E65A
$DC	LOG	$E941	$E9	RIGHT$	$E686
$DD	EXP	$EF09	$EA	MID$	$E691
$DE	COS	$EFEA			

The Function Entry Table is used during expression evaluation to obtain entry points to the function subroutines in the program area. It consists of 25 two byte entries with low-high addresses. The order of the entries corresponds to the tokens 210 to 234 assigned to the keywords SGN to MID$ as given in (Applesoft, Basic Programming Reference Manual, p.121). Table 2 gives the summary. The description of the function subroutines with their entry points are given in [Crossley].

The Operator Tag and Entry Table is used during expression evaluation. It consists of 10 three-byte entries corresponding to the tokens 200 to 209 assigned to the keywords + to × as given in (Applesoft, Basic Programming Reference Manual, p.121.) Of these three bytes, the first byte contains the Tag which also serves as a precedence number. The next two bytes contain the low-high addresses minus one of the entry points in the program area. Table 3 shows the Tag values and actual entry point addresses.

TABLE 3
Operator TAG and Entry Table From $D0B2-D0CF

Hex Token	Key Word	Hex Tag	Entry Point
$C8	+	$79	$E7C1
$C9	-	$79	$E7AA
$CA	*	$7B	$E982
$CB	/	$7B	$EA69
$CC	^	$7D	$EE97
$CD	AND	$50	$DF55
$CE	OR	$46	$DF4F
$CF	>	$7F	$EED0
$D0	=	$7F	$DE98
$D1	<	$64	$DF65

The Keyword Token Table is used by the Tokenizer routine which replaces keywords by appropriate tokens. It consists of the 107 keywords (from END to MID$) concatenated such that each byte is an ASCII character with high bit set to zero, unless the character is the last one of a keyword, in which case it is set to 1. e.g. it contains

EN**D**FO**R**NEX**T**...

where the bold character indicates that the high bit is one.

The ASCII Message Table contains ASCII characters where the individual message (e.g. the error message part "SYNTAX ERROR") is separated either by having the high bit set to its last character byte, or followed by a zero byte.

1.3 Scattered Data

Scattered data may occur in many places; some of them are the floating point constants, (see: [Crossley] and [Lingwood]), short table for high resolution graphics, (see: [Mesztenyi]).

1.4 Zero Page Load Data

The memory area $F10B-F126 is the CHRGET/CHRGOT routine followed by an initial random number which gets loaded into the Zero page $B1-CC during initialization.

1.5 Stored Program Area

Zero page location $67-68 contain the address (low-high) of the beginning of the stored program, usually $0801. From this address, the memory contains the tokenized label-statements ordered by their line numbers. The format of a tokenized label-statement is as follows:

2-byte	pointer (low-high address) to the next tokenized statement
2-byte	binary value (low-high) of the line number bytes of the tokenized compound-statement
n bytes	of the actual tokenized compound statement
1-byte	containing zero

The last tokenized labeled-statement is followed by two extra bytes containing zero. Thus the stored program has a chain of pointers starting with the contents of $67-68, and ending with a zero value. Each pointer indicates the beginning of a labeled-statement, while a byte containing zero indicates its end; three zero bytes indicate the end of the stored program.

1.6 Variable Areas

These areas and corresponding pointers are adequately described in (Applesoft, Basic Programming Reference Manual), with further explanations in [Golding].

2. CHRGET/CHRGOT SUBROUTINE.

The most important subroutine in Applesoft is the CHRGET/CHRGOT subroutine residing on the Zero page $B1-C8 with the TXTPTR imbedded at $B8-$B9. It has been described in [Crossley], but is repeated here because of its importance.

The CHRGOT entry ($B7) loads the register A with the contents of the memory whose address is in the TXTPTR ($B8-B9, low-high). CHRGET entry (B1) does the same except it increments the TXTPTR prior to loading. If the obtained byte is equal to the ASCII space ($20) then the control goes back to CHRGET, i.e. spaces (blanks) are skipped. Otherwise the flag Z is set if A=$3A or $00, i.e. ASCII colon (:) or null; flag C is set if A is not an ASCII number 0 to 9, i.e. A<$30 or A>$39; finally the control goes back to the calling routine.

The importance of this routine comes into light if one compares it to an instruction fetch cycle in a computer with the TXTPTR as a counter register. The instruction code is returned in register A, flags Z and C, ready to be executed (interpreted). The ASCII space code behaves like a no-op, and is automatically skipped. This feature is realized in the implementation of GOSUB-and RETURN-statements by placing the TXTPTR value together with line-number and tag $B0 on the stack in the GOSUB-statement, resetting them in the RETURN-statement.

Unfortunately, the CALL-statement has been implemented differently by not saving the above data in the stack. It would have been simple to implement in the same way as the GOSUB-statement, and the RETURN -statement could have served as a return address from the machine language subroutine. This would have allowed a call of the Applesoft routine at $D43C with a CALL-statement from a stored program with request for input of a compound-statement ending with RETURN ready to be executed in immediate mode, where the RETURN causes the return to the stored program.

3. PROGRAM STRUCTURE

The overall program structure of Applesoft can be illustrated by the following semantic program:

3.1. Initialization
3.2. Request and receive input from the keyboard.
3.3. Tokenize the input
3.4. If the first character of the input is an ASCII number then store the input as part of the stored program, and GOTO 3.2.
3.5. If the first character of the input is not an ASCII number then execute the input as a program, after which GOTO 3.2.

3.1 Initialization

The Initialization (starting at $F128) sets up the Zero page and various other pointers.

3.2 Input

The input request starts at $D43C. It uses the subroutine at $D52E to display the prompt symbol and through the Monitor GETLN, to receive the input line into the input buffer at $0200. It sets the high bits of the input data to zero, places a zero byte after the last input character, and initializes the TXTPTR to the input buffer address minus one.

3.3 Tokenization

The Tokenization Subroutine ($D559-D619, with entry at $D559) replaces the keywords with the appropriate tokens in the input buffer. It also removes blanks with the result still in the input buffer. It places two extra zero bytes at the end of the line. No syntax checking is performed by this routine.

Following Tokenization, the first character in the input buffer decides whether 3.4 or 3.5 is to be executed.

APPLESOFT INTERNAL STRUCTURE

3.4 Stored Program

If the first character in the input buffer is an ASCII number, then Applesoft assumes it to be the first character of a line-number of a labeled-statement and either inserts it or replaces an old labeled-statement with the same line-number in the stored program with the help of the routine starting at $D46A.

3.5 Execution

If the first character of the input is not an ASCII number, then Applesoft assumes the input to be a compound-statement ready to be executed. It sets the TXTPTR to the beginning of the input buffer and enters into an execution loop at $D805. At this stage, TXTPTR really behaves like a program counter. The execution of a statement advances or changes TXTPTR, e.g. to the stored program. Finally, the control returns to 3.2, requesting new input under the following condition:

(i) Execution of an end- or stop-statement
(ii) Encountering 3 consecutive zero bytes
(iii) Detecting syntax error without an onerr-statement.

Individual statements are recognized by their first, possibly tokenized, byte. If this is between $80 and $BF then it is assumed to be a token, and the statement is executed by jumping to the appropriate entry point listed in Table 1. Otherwise it is assumed to be a let-statement without the word LET. These statement execution routines are called subroutines, but not all of them return.

The execution loop in $D805-D848, and its preceding section in $D7D2-D894, is fairly complex. It is listed below with appropriate remarks.

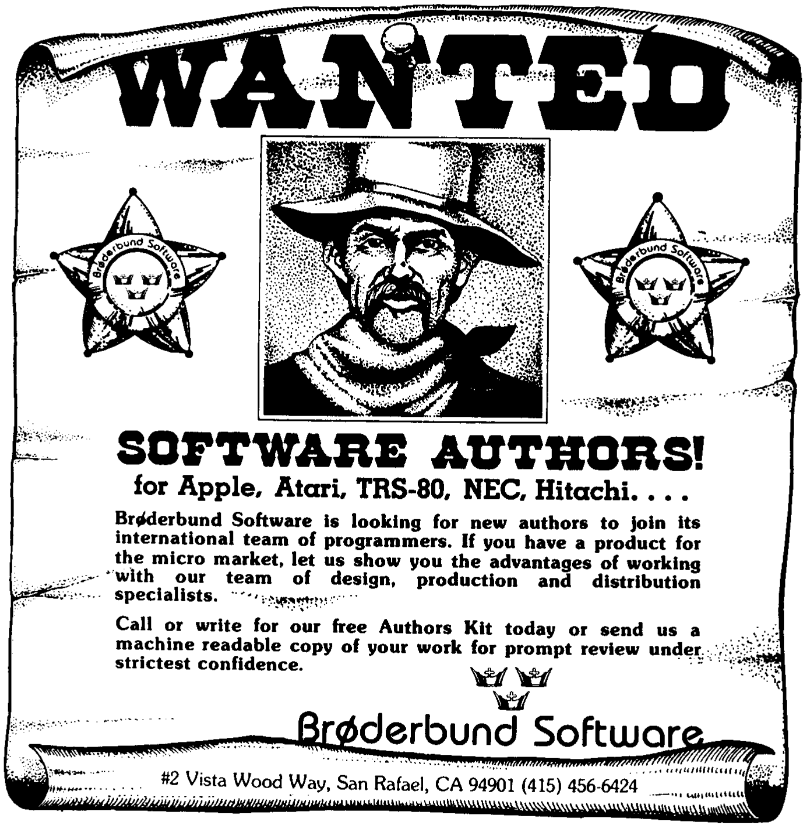
CONCLUSION

With the knowledge of the Data Structure, one may trace the internal workings of Applesoft based on the five point (3.1 to 3.5) Program Structure, and on the 64 statement interpreter subroutines with the given entry points. There are two difficult parts which need further documentation:

1. The expression evaluation routine, called by FRMEVL in [Crossley], which is used by many statement routines. I think that part of the complication is because Applesoft had been implemented before its syntactic rules were (correctly?) established.
2. The other difficulty lies in the multiple use of the stack. Beside the statement subroutines (GOSUB-, RETURN-,CALL-, FOR- and NEXT-statement), FRMEVL uses it, and also the internal program in Applesoft (JSR, RTS instructions).

References:

[1] Applesoft, Basic Programming Reference Manual.
[2]-[5] are all available in "Call-A.P.P.L.E. in Depth No.1." Apple Pugetsound Program Library Exchange. 1981.
[2] John Crossley: Applesoft Internals.
[3] C.K. Mesztenyi: Notes on Hi-Res Graphics Routines.
[4] David A. Lingwood: Amplifying Applesoft.
[5] Val J. Golding: Applesoft from Bottom to Top.
[6] William F. Luebbert: What's Where in the Apple. Micro Ink, Inc., Chelmsford, MA.



Statement Handler Routine $D7D2-$D804

```
NEWSTT          TSX             ;Save
                STX   $F8       ;Stackpointer
                JSR   $D858     ;Checks for Ctrl C
                LDA   $B8       ;Get
                LDY   $B9       ;TXTPTR
                LDX   $76       ;Check if immediate mode
                INX             ;($FF in current line nbr)
                BEQ   N1
                STA   $79       ;No, thus put TXTPTR into
                STY   $7A       ;Old  TXTPTR
N1              LDY   #$00      ;Check byte at TXTPTR
                LDA   ($B8),Y
                BNE   COLON     ;If non-zero then it should be ':'
                LDY   #$02      ;If zero then end of compound-st.
                LDA   ($B8),Y   ;Check for end of program
                CLC             ;Zero pointer 2 bytes further
                BEQ   PREND
                INY             ;It is a new labeled-statement
                LDA   ($B8),Y   ;Get and store new
                STA   $75       ;Current line nbr
                INY
                LDA   ($B8),Y
                STA   $76
                TYA             ;Update TXTPTR
                ADC   $B8
                STA   $B8
                BCC   EXECUTE
                INC   $B9
EXECUTE         BIT   $F2       ;Check for the trace bit
                BPL   L1        ;Notrace if positive
                LDX   $76       ;Trace is on, check
                INX             ;For mode
                BEQ   L1        ;No print in immediate mode
                LDA   #$23      ;Print out line nbr
                JSR   $DB5C     ;As trace information
                LDX   $75
                LDA   $76
                JSR   $ED24
                JSR   $DB57
L1              JSR   CHRGET    ;Get first byte of statement
                JSR   STYPE     ;Use JSR to get return address in
                                ;stack for
STTRET          JMP   NEWSTT    ;<--statement execution subroutine
                                ;returns here
PREND           BEQ   $D88A     ;End of program
STYPE           BEQ   $D857     ;Statement type check on its first byte
                SBC   #$80
                BCC   ASGST     ;<$80 then assign-statement
                CMP   #$40
                BCS   $D846     ;>$BF then error
                ASL             ;Otherwise get
                TAY             ;Entry point
                LDA   $D001,Y   ;From the 2-byte
                PHA             ;Statement-type table
                LDA   $D000,Y   ;And put it into stack
                PHA             ;As return address of CHRGET
                JMP   CHRGET    ;And go to there
ASGST           JMP   $DA46     ;Go to LET-st. routine
COLON           CMP   #$3A      ;Check for colon
                BEQ   EXECUTE   ;Yes, go to execute
                JMP   $DEC9     ;Otherwise error


Addresses:      NEWSST          $D7D2      N1     $D7E5
                EXECUTE         $D805      L1     $D81D
                STTRET          $D823      PREND  $D826
                STTYPE          $D828      ASGST  $D83F
                COLON           $D842
```