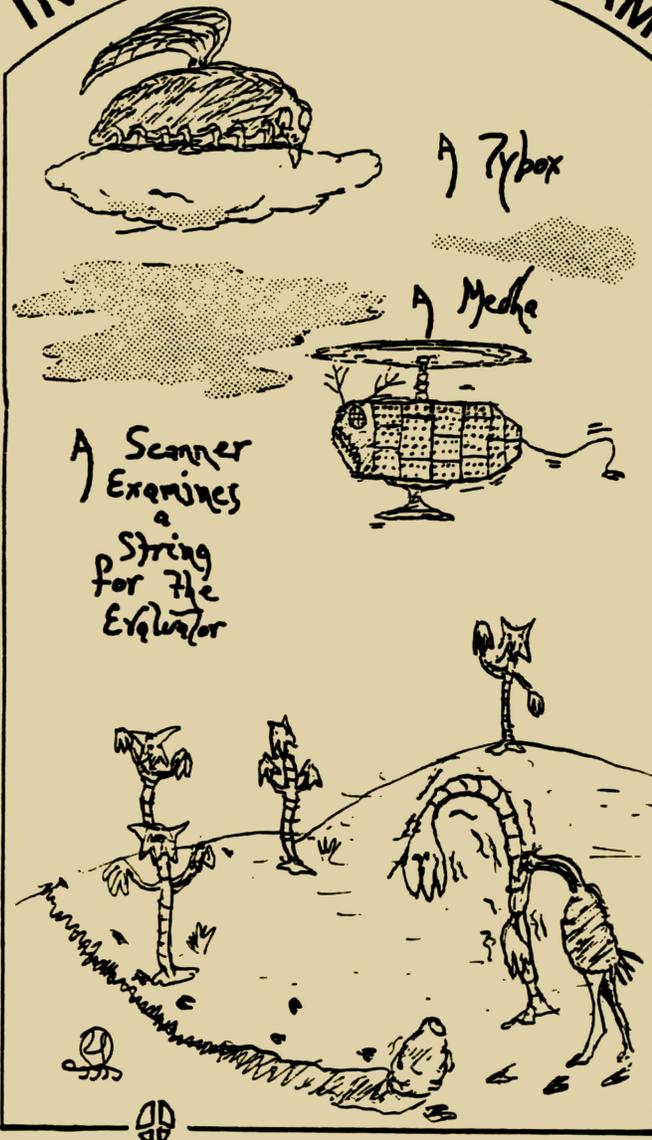


SAM76

the first LANGUAGE manual

IN THE LAND OF SAM



Second Edition

The SAM76 Language

The SAM76 language was designed by people for people - not by programmers for programmers. It follows a well defined syntax which is easy to learn and to read. The notation avoids the use of pseudo "English" words which are a frequent source of confusion and ambiguity in many of the other computer languages.

The SAM76 language can be used in as large a variety of tasks as one is able to imagine - this on personal computers without requiring computer specialists or programmers to intercede.

There are more than 150 functions - or instructions - available making the SAM76 language the most powerful available today, and it fits in approximately eight thousand bytes of memory; this can be ram or rom as the user desires.

The SAM76 language can be viewed as a real language which follows the user's stream of consciousness in much the same manner as spoken language. This permits the language in its written form as used by the computer and the user to serve as documentation.

The SAM76 language provides the user with the capability of requiring the computer to perform complex operations in many areas; a few of these are: Control, Text manipulation and editing, Simulation, Arithmetic with any desired precision.

The SAM76 language is interactive and reactive. As one task is accomplished the user continues and in effect the SAM76 language processor carries on a conversation, reacting to expressed desires.

The SAM76 language provides a uniquely flexible means to control facilities or to derive data from sources other than the user's keyboard.

The SAM76 language is a "string processor". This means that the units of information are not confined to any fixed length, but may be made up of any number of characters, or even no characters, as determined by the user. Entire strings may be manipulated by single commands.

The SAM76 language is interpretive. This means that when a string is evaluated and an expression found to contain an instruction or command, then the specified action is immediately performed and the resulting value, if any, replaces that expression in the string.

The SAM76 language facilitates the use of pre-defined procedures. This means that the user's procedures or scripts may be stored for potential use and later called by name and immediately acted upon, with variables supplied to specified arguments as part of the process.

The SAM76 language makes no distinction, except in the user's own use of information, between data and procedures. Procedures tell the processor what to do; data is the information acted upon by the procedures. Procedures may be modified when other procedures treat them as data.

The SAM76 language is most powerful in providing man-machine interaction permitting the user to modify his work and to intervene when desired. The language provides facilities to define and save scripts for subsequent use; this in effect can behave or operate as if they themselves were inherent functions of the language.

designed for you and your personal computer

SAM76 language reference

| | | | | | |
|--------------------------|-------------------------------|------------------------------|-------------------------------|-------------------------------|---|
| 238 - ef,s0 | wh# are Functions | : 149 - hp,t,d | How many Partitions | : 199 - sem,dev | Set "Echoplex" Mode active |
| 239 - en | wh# is processor ser. Number | : 114 - ht,t | Hide Text | : \sem,dev\ | "Echoplex" Mode inactive |
| 237 - et | wh# is processor Title | : \ht\ | Hide all Texts | : 222 - sf,f,t1,t2,...,t | Store File |
| 159 - ab,s1,s2,vt,vf | Alphabetic Branch | : 115 - ic | Input Character | : 157 - sfd,fun,dev | Specify Function Device |
| 128 - ad,n1,n2,n3,...,n | Add | : 116 - id,d | Input "D" characters | : 190 - sh,d,x | Shift the bits |
| 160 - ai,s0,s1,s2,...,s | Alphabetic Insertion | : 153 - idt,d | Input "D" Texts | : 253 - srn,n | Seed Random Number |
| 187 - and,x1,x2 | And the bits | : 136 - ig,d1,d2,vt,vf | If Greater | : 258 - sti,t1,t2,t3 | Set Time |
| 161 - as,s0,s1,s2,...,s | Alphabetic Sort | : 135 - ii,s1,s2,vt,vf | If Identical | : 129 - t,n1,n2,...,n | Subtract |
| 220 - bf,f,vz | Bring File | : 117 - im,s1,s2,...,s | Input to Match | : 231 - sw,s1,s2,s3,...,s | Switches |
| 113 - ca,s | Change Activator (current) | : 102 - ia,dev | Input String | : 232 - sy,s1,s2,...,s | System functions |
| \ca,s\ | Change Activator (initial) | : 152 - it | Input Text | : 127 - tb,t,vt,vf | Text Branch |
| 195 - cfc,d1,s | Change Fill Character schema | : 213 - iw,n | Input Wait | : 257 - tl,s1,s2 | Time |
| \cfc,d1,s\ | Change Fill Char. (initial) | : \lef,dev | Load External Function | : 125 - tm,d | Trace Mode activated |
| 193 - cIn,t1,d1,...,t,d | Change Id Number | : 216 - lf,s0,d1,...,d | List Files | : \tm\ | Trace Mode deactivated |
| 148 - cld,t | Characters Left of Divider | : \lr, ... | List Relationship | : 124 - tma | Trace Mode All activated |
| 191 - c11,d | Change Line Length (active) | : 105 - lt,s0,d1,d2,...,d | List Texts | : \tma\ | Trace Mode All deactivated |
| \c11,d\ | Change Line Length (initial) | : 214 - lw,s0,s1,s2,...,s | List Where | : 168 - tr,t,s | Trim |
| 133 - cnb,d | Change Number Base (active) | : 110 - mc,d | Multi-partition Character | : 218 - uf,f,t1,t2,...,t | Update File |
| \cnb,d\ | Change Number Base (initial) | : 146 - md,t,d | Move Divider to pos. "d" | : 169 - ut,cc | User Trap active |
| 266 - cpc,t1,d1,...,t,d | Change Protection Class | : \md,t,d\ | Move Divider "d" increments | : \ut\ | User Trap inactive |
| 147 - crd,t | Characters Right of Divider | : 109 - mt,t,s1,s2,...,s | Multi-part Text all matches | : 118 - vt,t1,t2,...,t | View Texts |
| 203 - cro,s1\ | Change Rub Out char. schema | : \mt,t,s1,s2,...,s\ | Multi-part Text next match | : 181 - wc,s1,s | Write Characters |
| \cro,s1\ | Change Rub Out (initial) | : 130 - mu,n1,n2,vz | Multiply | : 175 - wi,xn1,yn1 | Write Initialise |
| 132 - ct,t1,t2,t3,...,t | Combine Texts (superseding) | : 111 - ni,vt,vf | Neutral Implied | : 179 - wi | Width Left |
| \ct,t1,t2,t3,...,t\ | Combine Texts (save current) | : 188 - not,x | Not (complement) the bits | : 178 - wr | Width Right |
| 250 - cac,s1 | Change Warning Character | : 209 - nu,s1,s2,...,s | Null | : 180 - ws,xn1,yn1,...,xn,yn | Write Straight Lines |
| \cac,s1\ | Change Warn. Char. (initial) | : 246 - oj,s,s1,d,s2 | Output Justified lines | : 176 - wx | Write "X" displacement |
| 261 - cws,d | Change Work Space | : 248 - op,s,s1,d,s2 | Output Padded lines | : 177 - wy | Write "Y" displacement |
| \cws,d\ | Character to "X" | : 186 - or,x1,x2 | Or the bits | : 170 - xc,x1,x2,...,x | "X" to Character |
| 171 - cx,s0,s | Change "X" Base (active) | : 101 - os,s | Output String | : 271 - xcf,s,x | eXperimental Change Function |
| 200 - cwb,d | Change "X" Base (initial) | : 154 - ot,t1,t2,...,t | Output Texts | : 172 - xd,x | "X" to Decimal |
| \cwb,d\ | Date | : 108 - pc,d | Partition Character | : 255 - xi,port1 | eXperimental Input |
| 259 - da,s0 | Divide | : 174 - pl,s1,s2,...,s | Plot | : 123 - xj,x | eXperimental Jump |
| 131 - di,n1,n2,vz | Define Quote | : 162 - ps,d,s1,s2 | Pad String | : 256 - xo,x,port | eXperimental Output |
| 208 - dq,s | Define Relationship | : 107 - pt,t,s1,s2,...,s | Partition Text all matches | : 270 - xqf,s | eXperimental Query Function |
| - dr,t,a,o,v | Define String | : \pt,t,s1,s2,...,s\ | Partition Text next match | : 119 - xr,x | eXamine Register |
| 164 - ds,d,s | Duplicate Text (superseding) | : 196 - qfc,s0 | Query Fill Character schema | : 121 - xrp,x | eXamine Register Pair |
| 103 - dt,t,s,d1,d2 | Define Text (superseding) | : 194 - qin,s0,t1,t2,...,t | Query Id Number | : 120 - xw,x1,x2 | eXperimental Write in reg. |
| \dt,t,s,d1,d2\ | Define Text (save current) | : 197 - qid,t | Query Left of Divider | : 122 - xwp,x1,x2 | eXperimental Write req. Pair |
| 173 - dx,d,x | Decimal to "X" | : 192 - qll | Query Line Length | : 126 - yt,t,s,vt,vf | Ya There |
| 206 - ea,t1,t2,...,t | Erase All excepting | : 134 - qnb | Query Number Base | : 182 - zd,r,v-,v0,v+ | "Z" reg. Decrement and branch |
| 207 - ed,t,d1,d2,vz | Extract "D" characters | : 202 - qof | Query Over Flow conditions | : 183 - zl,r,v-,v0,v+ | "Z" reg. Increment and branch |
| 224 - ef,f1,f2,...,f | Erase Files | : 167 - qp,t | Query Partition | : 184 - zq,r | "Z" reg. Query |
| 151 - ep,t,p1,p2,...,p | Erase Partitions | : 267 - qpc,s0,t1,t2,...,t | Query Protection Class | : 185 - zs,r,n | "Z" reg. Set |
| - er, ... | Express Relationship | : 198 - qrd,t | Query Right of Divider | | |
| 104 - et,t1,t2,...,t | Erase Text | : 204 - qrol | Query Rub Out char. schema | | |
| \et,t1,t2,...,t\ | Erase all occurrences of Text | : 205 - qta | Query Text Area used | | |
| 249 - etb,s | Erase Trailing Blanks | : 251 - qwc,a2,a1,...,a | Query Warning Characters | | |
| 112 - ex,f | Exit | : 262 - qws | Query Work Space | | |
| 226 - fb,f,vt,vf | File Branch | : \qws\ | | | |
| 137 - fc,t,vz | Fetch Character | : 201 - qxb | Query "X" Base | | |
| 138 - fdc,t,d,vz | Fetch "D" Characters | : 215 - ra,d,s1,s2,s3,...,s | Return Argument | | |
| 139 - fde,t,d,vz | Fetch "D" Elements | : 263 - rcp,d1,d2,s | Return Character Picture | | |
| 140 - fde,t,d,s,vz | Fetch "D" Matches | : 166 - ri | Restart Initialized | | |
| 141 - fe,t,vz | Fetch Element | : 245 - rj,s,s1,d,s2 | Return Justified lines | | |
| 142 - ff,t,d,vz | Fetch Field | : 252 - rn,n | Random Number | | |
| 143 - fl,t,s,vz | Fetch Left match | : 189 - rot,d,x | Rotate the bits | | |
| 145 - fp,t,x1,...,x | Fetch Partition | : 247 - rp,s,s1,d,s2 | Return Padded lines | | |
| 144 - fr,t,s,vz | Fetch Right match | : 165 - rr,s | Return to Restart | | |
| 106 - ft,t,s1,s2,...,s | Fetch Text | : 163 - rs,s | Reverse String | | |
| 210 - ftb,t,s,vz | Fetch To Break character | : 228 - saf,dev | Select All File function dev. | | |
| 211 - fts,t,s,vz | Fetch To Span character | : 158 - sar | "Auto Return" on line feed | | |
| 212 - hc,s | How many Characters | : \sar\ | no Auto Return on line feed | | |
| 150 - hm,t,s | How many Matches | : 260 - sda,da,mo,yr | Set Date | | |
| | | | | | Expression formats, legend, syntax and conventions: |
| | | | | function,arguments,... | Active Expression |
| | | | | \function,arguments,...\ | Neutral Expression |
| | | | | x,x1,.. | "x" base numbers - f file name |
| | | | | d,d1,.. | Decimal numbers - t text name |
| | | | | n,n1,.. | "n" base numbers - vz default value |
| | | | | s0 | prefixing string - v-,v+,v0 conditional value |
| | | | | s,s1,.. | character strings - vt,vf true/false value |
| | | | | | Protection syntax - l..../ (....) <....> @char. |
| | | | | | Active syntax - S: \fn,arguments/ - M: \fn,arguments: |
| | | | | | Neutral syntax - S: sfn,arguments/ - M: \fn,arguments; |
| | | | | | @vt,t/= partition [d], multi-partition [M], divider ["] |
| | | | | | <ace-xxx> special condition encountered |
| | | | | | <nav-xxx> xxx not available |

%os,%is// is the Restart Expression which is originally loaded. It means: "output that string which results from the evaluation or execution of the string to be input". Thus:

1. Input a String
2. Evaluate said string
3. Output the result of the evaluation

In the examples that follow, the "os" of the Restart Expression will not be shown, but its presence is implied. For clarity in these examples output will be shown between a pair of curly braces thus: { ... }.

```
ABCDEF GH=(ABCDEF GH)
```

The "os" of the Restart Expression causes to be output that string which was entered through execution of the "is" (Input String) of the Restart Expression. The "=" equal sign is the Activator, signalling the end of the input string.

```
%os,APPLE/=(APPLE)
```

The function "os" (output string) in the expression causes the output of the second argument of the expression; the comma is sensed as a delimiter between arguments and only the second argument will be output by the "os" function.

```
%os,APPLE<,>ORANGE/=(APPLE,ORANGE)
%os,<APPLE,ORANGE>/=(APPLE,ORANGE)
%os,APPLE@,ORANGE/=(APPLE,ORANGE)
```

Here the comma is protected, hence it does not act as a delimiter, and is entered as part of the input string. As part of the string it is output by the "os" function. Note that the protective symbol pair (in this case <...>) may be anywhere as long as the comma is enclosed. Other protective symbol pairs that may be used are (...) and !.../; in addition any single character immediately preceded by a "@" sign is also protected as shown on the third line example.

```
%dt,A,APPLE@,ORANGES/ =
```

Define a Text named A with contents APPLES,ORANGES and store it in a section of memory named the "Text Area".

```
%os,%ft,A/=(APPLES)
%os,%A/=(APPLES)
%os,%ft,A/=(APPLES,ORANGES)
%os,%A/=(APPLES)
```

Fetch from the Text Area "A" and output its contents. If the name of the text is not the same as that of any of the functions of the language, the fetch may be made as shown on the second line of the example; this is said to be an "implied fetch". Should the text contain symbols which should normally have been protected, or if it is desired not to evaluate the text to be fetched, then the format of the third line should be used; this is said to be a "neutral explicit fetch". The fourth line shows a "neutral implied fetch"; this behaves in a manner that is identical to the first two lines of the example, but information is retained in the computer that it was a "neutral implied" fetch.

```
%A/=(APPLES)
%ft,A/=(APPLES,ORANGES)
```

Fetch the text named A, both actively and explicitly neutrally. Output is effected by the "os" function of the Restart Expression as indicated in the following sequence:

1. %os,%is//
2. %os,%A//
3. %os,APPLES,ORANGES/
4. APPLES

```
%dt,A,THE DOG AND THE CAT AND THE HORSE/ =
```

As a part of defining this text named A, the previously defined text also named A is erased from the Text Area, and the new text A, containing the new text string is created.

```
%dt,B,%A/%ct,C,A/ =
%os,%A/=(THE DOG AND THE CAT AND THE HORSE)
%os,%B/=(THE DOG AND THE CAT AND THE HORSE)
%os,%C/=(THE DOG AND THE CAT AND THE HORSE)
```

Define a text named B as the value resulting from fetching A and create C by copying A using the "ct" copy text function.

```
%pt,B,THE DOG,AND,CAT,HORSE/ =
```

Partition the text named B on the character patterns, "THE", "DOG", "AND", "CAT", "HORSE", creating partitions at those locations in Text B where each pattern appears. The partitions where the first pattern occurred are given a value of [1], the partitions where the second pattern occurred are given value [2], etc.

```
%vt,B/=[(1) (2) (3) (1) (4) (3) (1) (5)]
```

"vt" (View Text) will show the numerical value and location of the partitions in a Text. Note that the unpartitioned patterns (the spaces between the words) remain intact.

```
%B,LE,CHIEN,ET,CHAT,CHEVAL/
=(LE CHIEN ET LE CHAT ET LE CHEVAL)
```

The partitions with values 1, 2, 3 etc., are plugged by the second, third, fourth etc. arguments of the fetch of Text B, and the plugged string resulting is then output by the Restart Expression. A new line code was input before the Activator. This is why the output is on the second line.

```
%vt,B/=[(1) (2) (3) (1) (4) (3) (1) (5)]
```

Note that Text B still has the partitions.

```
%dt,B,%B,LE,CHIEN,ET,CHAT,CHEVAL/ =
%B/=(LE CHIEN ET LE CHAT ET LE CHEVAL)
%A/=(THE DOG AND THE CAT AND THE HORSE)
%ft,*/=(%A*C*B)
%ft,
/ =|
A
C
B)
```

This will redefine B, plugging the partitions as indicated; note that any unplugged partitions at this point would be plugged with "null" strings. The text B, had been defined as the same as text A. Then it was partitioned on the English words in it and was then redefined with the corresponding French words replacing the English ones.

The names of the Texts in the Text Area are determined through use of the "lt" (List Texts) function. Each text name is PRECEDED by whatever delimiting character pattern the user specifies as the second argument of the expression. One example uses an asterisk, and the other example has a new line code as the second argument of the expression.

```
%dt,A,%os,THIS IS A PROCEDURE/// =
%A/=(THIS IS A PROCEDURE)
%ft,A/=(%os,THIS IS A PROCEDURE/)
```

A procedure is a text consisting of one or more expressions executed by fetching said text "actively". An explicit neutral fetch serves only to fetch it without its being executed. The protective pair !.../ serves to prevent execution during the process of definition. Partitions, if any may be plugged during the fetching process at the time of execution. Other examples of procedures follow.

```
%dt,SQUARE,%mu,*,%/// =
%pt,SQUARE,*/ =
%vt,SQUARE/=(%mu,(1),(1)/)
% SQUARE,9/=(81)
% SQUARE,12/=(144)
```

```
%dt,HONDY,%os,
WHAT IS YOUR NAME?- /%os,
WELL HELLO THERE %is//// =
%SHOWDY/ =
(WHAT IS YOUR NAME?- )BILL =
(WELL HELLO THERE BILL)
```

As strings are evaluated from the inside out and from left to right, procedures can be nested within other procedures. In this case the Activator must be entered after the name (BILL in this case), to signify the end of the "is" function. This value "BILL", then replaces the %is/ in the procedure and is output by the second "os".

```
%dt,LOOP,%os,
THIS PROCEDURE LOOPS/%LOOP/// =
%LOOP/ =|
THIS PROCEDURE LOOPS
THIS PROCEDURE LOOPS
THIS PROCEDURE LOOPS
THIS PROCED
<sc-e-os>|
```

To make a procedure loop, it must fetch itself. If the looping procedure has partitions in it, they will be plugged during the fetching process. In such cases if no plugs are specified, null strings will be used. In this example the loop was broken from the keyboard by hitting the "rubout" or "del" key; the <sc-e-os> message means "special condition encountered" during the execution of "os".

```
%dt,F,%ii,*1,1,
%mu,*%F,%su,*1//////// =
%pt,F,*/ =
%F,1/=(1)
%F,3/=(6)
%F,5/=(120)
```

A short recursive procedure may find the factorial of any number. This procedure tests the entered number, plugging the partitions, to see if it is a 1; if not, the factorial of the entered number is that number multiplied by the factorial of that number minus 1, which is computed by fetching F. Sometimes it is desired to organize the procedures in several lines, or use tabs to indent the lines; these formatting characters (used only for esthetic reasons) are not really part of the executable matter, and would clutter up the scanning process. Such clutter is avoided by preceding characters which have only an aesthetic meaning with the " " or "grave" accent mark.

SAM76 Inc.
Box 257, R.R.1
Pennington, N. J., 08534, U.S.A.

-
Available materials and prices effective Apr 1, 1981

Printed Matter:

SAM76 Language Manual - 240 pps. \$20.00
SAM76 Beginners Tutorial booklet \$ 5.00

Machine Readable materials:

SAM76 Distribution Disk - \$25.00

Contains object code for 8080 and Z80
Source code for graphics functions,
real time clock and some miscellaneous
other functions; variety of applications
scripts, and demonstrations as well as
miscellaneous tutorial material.
Available formats (see NOTE A,B,C,D,E,F)

SAM76 TRS80 disk, includes brochure \$25.00

SAM76 Complete 8080/Z80 Source Code \$60.00
Requires Z80 C.D.L. Macro assembler (Note A)

SAM76 Complete DEC10 system, includes \$500.00
source code, Hershey fonts and 5 manuals

SAM76 Adventure Game (NOTE: A,B,D,E1) \$25.00

SAM76 "Hershey" graphic incremental vector tables
I - Occidental Fonts excluding Gothic \$25.00
II - Gothic Fonts and Oriental Index \$25.00
III - Oriental Fonts \$25.00
Set of three disks \$60.00
Available formats are "A,B,C,D,E"

Notes:

A - 8inch single density CP/M format
B - 5 1/4 inch - Micropolis MOD II CP/M
C - 5 1/4 inch - TRS-80 CP/M or TP/M
D - 5 1/4 inch - TRS-80 standalone
E1 - 5 1/4 inch - APPLE Z80 CP/M
E2 - 5 1/4 inch - APPLE Z80 CP/M GRAPHICS
F - 5 1/4 inch - North Star Double Density

For other formats contact SAM76 by phone (609) 466-1129

TERMS: All above prices are net - payment with order
cash, money order or personal check - no credit
cards. Prices include shipping postpaid fourth
class for printed matter unless disks included
in which case first class. Add \$2.00 for first
class for books alone. Overseas add \$7.00 for
AIR book rate.

Net 30 prices are those quoted above multiplied
by two. One copy of invoice furnished gratis,
on request, if additional copies are required
add \$5.00 for each desired copy.