SAM76 language reference

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

Expression formats, legend, syntax and conventions:

```
|function,arguments,..,.|      Active Expression
\function,arguments,..,.\      Neutral Expression

x,xl,..  "x" base numbers  - f        file name
d,dl,..  Decimal numbers   - t        text name
n,nl,..  "n" base numbers  - vz       default value
s0       prefixing string  - v-,v+,v0 conditional value
s,sl,..  character strings - vt,vf    true/false value

Protection syntax - |..../   (....)   <....>     @char.
Active syntax   - S: $fn,arguments/  - M: |fn,arguments;
Neutral syntax  - S: &fn,arguments/  - M: |fn,arguments;

$vt,t/=  partition [d], multi-partition [$d], divider [^]
<sce-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: { ... }.

```
ABCDEFGH={ABCDEFGH}
```

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 enfolded. 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,APPLES@,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}
%lt,*/={*A*C*B}
%lt,
/={
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.

SAM76 Inc
Box 257, RR1. PENNINGTON NJ
08534

```
%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,HOWDY,!%os,
WHAT IS YOUR NAME?- /%os,
WELL HELLO THERE %is////=
%HOWDY/=
{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
<sce-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 <sce-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.