


```

001 PLL1013 0001 0 MODULE SMG$SPUT_TEXT_TO_BUFFER ( %TITLE 'Put text to display buffer'
3-1 0002 0 -IDENT = '1-013' ! File: SMGPUTTEX.B32 Edit: PLL1013
0003 0 ) =
0004 1 BEGIN
0005 1
0006 1 *****
0007 1 *
0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY *
0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS. *
0010 1 * ALL RIGHTS RESERVED. *
0011 1 *
0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *
0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE *
0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER *
0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *
0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY *
0017 1 * TRANSFERRED. *
0018 1 *
0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE *
0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT *
0021 1 * CORPORATION. *
0022 1 *
0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS *
0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL. *
0025 1 *
0026 1 *
0027 1 *****
0028 1
0029 1
0030 1 ++
0031 1 FACILITY: Screen Management
0032 1
0033 1 ABSTRACT:
0034 1
0035 1 This is an internal routine used by screen management procedures to
0036 1 place user's text into a display buffer. The text is spanned for
0037 1 special characters.
0038 1
0039 1 ENVIRONMENT: User mode - AST reentrant
0040 1
0041 1 AUTHOR: P. Levesque, CREATION DATE: 14-Apr-1983
0042 1
0043 1 MODIFIED BY:
0044 1
0045 1 1-001 - Original. PLL 14-Apr-1983
0046 1 1-002 - Finish coding. PLL 20-Apr-1983
0047 1 1-003 - Add error message, character set buffer allocation. PLL 4-May-1983
0048 1 1-004 - Fix second half of the scan table to agree with actions for
0049 1 DEC Multinational. PLL 5-May-1983
0050 1 1-005 - If on the last line and we have found a line feed, scroll. PLL 11-May-1983
0051 1 1-006 - If a bell character is found, call SMG$RING_BELL instead of setting
0052 1 a bell bit. PLL 20-May-1983
0053 1 1-007 - If a LF is found, scroll according to the new dcb top & bottom of
0054 1 scrolling region fields. PLL 26-May-1983
0055 1 1-008 - If an ESC is detected, call the terminal simulator routine to
0056 1 interpret the sequence and perform the correct SMGS function.
0057 1 PLL 7-Jul-1983
    
```

:	58	0058	1	:	1-009 - Allow 2 'reserved' positions in upper half of table to pass thru
:	59	0059	1	:	as printable characters. PLL 17-Aug-1983
:	60	0060	1	:	1-010 - SMG\$SIM_TERM may set the graphics bit in the DCB's default
:	61	0061	1	:	attributes byte. Take this into account when copying the attribute
:	62	0062	1	:	bytes for characters into the buffer. PLL 29-Aug-1983
:	63	0063	1	:	1-011 - Call SMG\$SIM_TERM when DCB_V_ALLOW_ESC is set. PLL 2-Sept-1983
:	64	0064	1	:	1-012 - In order to print carriage control characters instead of execute
:	65	0065	1	:	them, check the DCB_V_DISPLAY_CONTROLS bit and move the ascii rep
:	66	0066	1	:	into the text buffer in a different way. PLL 23-Sep-1983
:	001 PLL1013	0067	1	:	1-013 - Remove dot on DCB argument in SMG\$RING_BELL call. PLL 10-Oct-1984
:	67	0068	1	:	--
:	68	0069	1	:	


```

: 70      0070 1 %SBTTL 'Declarations'
: 71      0071 1
: 72      0072 1 | SWITCHES:
: 73      0073 1 |
: 74      0074 1 |
: 75      0075 1 SWITCHES ADDRESSING_MODE (EXTERNAL = GENERAL, NONEXTERNAL = WORD_RELATIVE);
: 76      0076 1
: 77      0077 1 |
: 78      0078 1 | LINKAGES:
: 79      0079 1 |
: 80      0080 1 |     NONE
: 81      0081 1 |
: 82      0082 1 | TABLE OF CONTENTS:
: 83      0083 1 |
: 84      0084 1 |
: 85      0085 1 FORWARD ROUTINE
: 86      0086 1     SMG$$PUT_TEXT_TO_BUFFER;
: 87      0087 1
: 88      0088 1 |
: 89      0089 1 | INCLUDE FILES:
: 90      0090 1 |
: 91      0091 1 |
: 92      0092 1 REQUIRE 'RTLIN:SMGPROLOG';           ! defines Psects, macros, data base
: 93      0170 1
: 94      0171 1 |
: 95      0172 1 | MACROS:
: 96      0173 1 |
: 97      0174 1 |     NONE
: 98      0175 1 |
: 99      0176 1 | EQUATED SYMBOLS:
100     0177 1 |
101     0178 1 |     NONE
102     0179 1 |
103     0180 1 | FIELDS:
104     0181 1 |
105     0182 1 |     NONE
106     0183 1 |
107     0184 1 | PSECTS:
108     0185 1 |
109     0186 1 |
110     0187 1 |
111     0188 1 | EXTERNAL REFERENCES:
112     0189 1 |
113     0190 1 |
114     0191 1 EXTERNAL ROUTINE
115     0192 1     SMG$$SIM_TERM,
116     0193 1     SMG$$SCROLL_AREA,
117     0194 1     SMG$RING_BECL;
118     0195 1 |
119     0196 1 EXTERNAL LITERAL
120     0197 1     SMG$_FATERRLIB,
121     0198 1     SMG$_STRTERESC;
122     0199 1 |
123     0200 1 ! Some constants needed by reference.
124     0201 1 OWN
125     0202 1     ALLONES      : BYTE INITIAL (-1);
: 126     0203 1
  
```

```

: 127      0204 1 ! The following macro is used to move a control character into the
: 128      0205 1 ! text buffer in such a way that output will later convert to the
: 129      0206 1 ! appropriate device dependent graphic character.
: 130      0207 1
: 131      0208 1 MACRO
: 132      M 0209 1 $INSERT_CTRL_CHAR (CHAR) =
: 133      M 0210 1 BEGIN
: 134      M 0211 1 LOCAL
: 135      M 0212 1 INDEX,
: 136      M 0213 1 REMAINING_COLS;
: 137      M 0214 1
: 138      M 0215 1 REMAINING_COLS = .DCB [DCB_W_NO_COLS] - .DCB [DCB_W_CURSOR_ROW];
: 139      M 0216 1 INDEX = $$SMG$LINEAR (.DCB [DCB_W_CURSOR_ROW], .DCB [DCB_W_CURSOR_COL]);
: 140      M 0217 1
: 141      M 0218 1 IF 1 GTR .REMAINING_COLS
: 142      M 0219 1 THEN
: 143      M 0220 1 WORK_OVERFLOW = .BYTES_REMAINING
: 144      M 0221 1 ELSE
: 145      M 0222 1 BEGIN ! move the low nibble into the high nibble
: 146      M 0223 1 LOCAL
: 147      M 0224 1 SHIFT_NIBBLE : BYTE,
: 148      M 0225 1 WORK_ATTR;
: 149      M 0226 1 SHIFT_NIBBLE = (CHAR <0,4>) ^ 4;
: 150      M 0227 1 CHSMOVE (1, SHIFT_NIBBLE, TEXT_BUF [.INDEX]);
: 151      M 0228 1 WORK_ATTR = ATTR_M_USER_GRAPHIC OR .ATTR_CODE;
: 152      M 0229 1 CHSMOVE (1, WORK_ATTR, ATTR_BUF [.INDEX]);
: 153      M 0230 1 END;
: 154      M 0231 1
: 155      M 0232 1 DCB [DCB_W_CURSOR_COL] = .DCB [DCB_W_CURSOR_COL] + 1;
: 156      M 0233 1 IF .DCB [DCB_W_CURSOR_COL] EQL .DCB [DCB_W_NO_COLS]
: 157      M 0234 1 THEN
: 158      M 0235 1 DCB [DCB_W_CURSOR_COL] = .DCB [DCB_W_NO_COLS];
: 159      0236 1 ENDX;
: 160      0237 1
: 161      0238 1 !<BLF/PAGE>

```

```

163 0239 1
164 0240 1
165 0241 1
166 0242 1
167 0243 1
168 0244 1
169 0245 1
170 0246 1
171 0247 1
172 0248 1
173 0249 1
174 0250 1
175 0251 1
176 0252 1
177 0253 1
178 0254 1
179 0255 1
180 0256 1
181 0257 1
182 0258 1
183 0259 1
184 0260 1
185 0261 1
186 0262 1
187 0263 1
188 0264 1
189 0265 1
190 0266 1
191 0267 1
192 0268 1
193 0269 1
194 0270 1
195 0271 1
196 0272 1
197 0273 1
198 0274 1
199 0275 1
200 0276 1
201 0277 1
202 0278 1
203 0279 1
204 0280 1
205 0281 1
206 0282 1
207 0283 1
208 0284 1
209 0285 1
210 0286 1
211 0287 1
212 0288 1
213 0289 1
214 0290 1
215 0291 1
216 0292 1
217 0293 1
218 0294 1
219 0295 1

```

The table below (CHAR TABLE) is used with a SCANC instruction to detect characters that have an impact on how text needs to be positioned in a text buffer that models what is on a portion of the screen. Each character position is occupied by a code indicating the kind of action that this character has on text placement. Characters are grouped into 10 categories based on their impact on the terminal and hence on their impact on what should be placed in the buffer at what position.

These categories (codes) are:

Action Code	Action
-----	-----
0	Normal processing. Character occupies next available slot in buffer. Cursor column is advanced by 1 after placement.
1	Character can be discarded. Cursor is not advanced.
2	Character can be discarded. Cursor is not modified, but a note must be made that the bell needs to be sounded.
3	Character can be discarded, but cursor must be backed up one column. Be careful about cursor already being in column 1.
4	Character can be discarded, but cursor must be advanced to next TAB stop and intervening character positions in the buffer are undisturbed.
	TAB stops are assumed to be set in the following columns with column numbering starting at 1: 9, 17, 25, 33, 41, 49, 57, 65, 73 (width=80)
	9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105, 113, 121, 129 (width=132)
5	Character can be discarded. Cursor must be advanced by one line.
6	Character can be discarded. Cursor must be advanced by one line. (VT treated the same as #5, FF.)
7	Character can be discarded. Effect is to clear the buffer and reset the cursor to line 1 column 1.
8	Character can be discarded. Effect is to set cursor to column 1 of current line.
9	Character can be discarded. For this version, ESC terminates the string. Eventually, subsequent


```

: 220 0296 1 :
: 221 0297 1 :
: 222 0298 1 :
: 223 0299 1 :
: 224 0300 1 :
: 225 0301 1 :
: 226 0302 1 :
: 227 0303 1 :
: 228 0304 1 :
: 229 0305 1 :
: 230 0306 1 :
: 231 0307 1 :
: 232 0308 1 :
: 233 0309 1 :
: 234 0310 1 :
: 235 0311 1 :
: 236 0312 1 :
: 237 0313 1 :
: 238 0314 1 :
: 239 0315 1 :
: 240 0316 1 :
: 241 0317 1 :
: 242 0318 1 :
: 243 0319 1 :
: 244 0320 1 :
: 245 0321 1 :
: 246 0322 1 :
: 247 0323 1 :
: 248 0324 1 :
: 249 0325 1 :
: 250 0326 1 :
: 251 0327 1 :
: 252 0328 1 :
: 253 0329 1 :
: 254 0330 1 :
: 255 0331 1 :
: 256 0332 1 :
: 257 0333 1 :
: 258 0334 1 :
: 259 0335 1 :
: 260 0336 1 :
: 261 0337 1 :
: 262 0338 1 :

```

characters need to be inspected to see if they constitute a recognized escape sequence whose effect must be simulated-- E.g., cursor setting, rendition setting.

Some problems with this are:

1. What to do about sequences that we don't recognize ?
2. What to do about sequences that we recognize as ones that can cause confusion later is allowed to be sent to terminal -- E.g. select graphics rendition, etc ?

10

Character can be discarded. Character is treated as a no-op. It is broken out separately in case we ever need to do something special with it.

In summary:

Hex Character Codes	ASCII Character	Action Code
00 to 06	NUL to ACK	1
07	BEL	2
08	BS	3
09	HT	4
0A	LF	5
0B	VT	6
0C	FF	7
0D	CR	8
0E to 0F	SO to SI	9
10 to 1A	DLE to SUB	1
1B	ESC	9
1C to 1F	FS to US	1
20 to 7E	SP to -	0
7F	DEL	10
80 to 9F	control chars	1
A0	reserved	1
A1 to FE	printing chars	0
FF	reserved	1

```
.. 264 0339 1 GLOBAL
.. 265 0340 1 CHAR_TABLE : VECTOR [256, BYTE] INITIAL ( BYTE (
.. 266 0341 1 1st half is US ASCII
.. 267 0342 1 for DEC Multinational set (default)
.. 268 0343 1 1. 1. 1. 1. 1. 1. 1. 2. 3. 4. 5. 6. 7. 8. 9. 9. 00 to 0F
.. 269 0344 1 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 9. 1. 1. 1. 1. 10 to 1F
.. 270 0345 1 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 20 to 2F
.. 271 0346 1 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 30 to 3F
.. 272 0347 1 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 40 to 4F
.. 273 0348 1 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 50 to 5F
.. 274 0349 1 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 60 to 6F
.. 275 0350 1 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 10. 0. 70 to 7F
.. 276 0351 1 2nd half is DEC Supplemental Graphics
.. 277 0352 1 for DEC Multinational set (default)
.. 278 0353 1 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 80 to 8F
.. 279 0354 1 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 90 to 9F
.. 280 0355 1 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. A0 to AF
.. 281 0356 1 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. B0 to BF
.. 282 0357 1 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. C0 to CF
.. 283 0358 1 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. D0 to DF
.. 284 0359 1 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. E0 to EF
.. 285 0360 1 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. F0 to FF
.. 286 0361 1 )):
.. 287 0362 1
.. 288 0363 1
.. 289 0364 1
.. 290 0365 1
.. 291 0366 1 !<BLF/PAGE>
```



```

293 0367 1 ZSBTTL 'SMGSSPUT TEXT TO BUFFER - Put text to buffer'
294 0368 1 GLOBAL ROUTINE SMGSSPUT_TEXT_TO_BUFFER (
295 0369 1     DCB : REF BLOCK [,BYTE],
296 0370 1     ATTR_CODE : BYTE,
297 0371 1     TEXT_LEN,
298 0372 1     TEXT_ADDR,
299 0373 1     CHAR_SET,
300 0374 1     OVERFLOW
301 0375 1 ) =
302 0376 1
303 0377 1 ++
304 0378 1 FUNCTIONAL DESCRIPTION:
305 0379 1     This procedure places a text string into a buffer given the
306 0380 1     current row and column in the buffer where output is to go.
307 0381 1     The input text string is scanned for special characters that
308 0382 1     prohibit simply moving the text into the buffer. For example,
309 0383 1     TABs reposition the maintained cursor position and the text
310 0384 1     must be deposited at the appropriate tab boundaries as a
311 0385 1     function of current position in the line. Escape sequences
312 0386 1     are not handled; an escape character is treated as a terminator,
313 0387 1     and a qualified success status will be returned to indicate
314 0388 1     that truncation occurred.
315 0389 1
316 0390 1     Positions in BUFFER that are modified have the corresponding
317 0391 1     positions in ATTR_BUFFER and CHAR_BUFFER set.
318 0392 1
319 0393 1
320 0394 1
321 0395 1 CALLING SEQUENCE:
322 0396 1
323 0397 1     ret_status.wlc.v = SMGSSPUT_TEXT_TO_BUFFER (
324 0398 1         DCB.mab.r,
325 0399 1         ATTR_CODE.rb.v,
326 0400 1         TEXT_LEN.rl.v,
327 0401 1         TEXT_ADDR.rl.v,
328 0402 1         CHAR_SET.rl.v
329 0403 1         [,OVERFLOW.wl.r])
330 0404 1
331 0405 1 FORMAL PARAMETERS:
332 0406 1
333 0407 1     DCB.mab.r     Address of virtual display control block.
334 0408 1                 Various fields from within in this block are
335 0409 1                 are interrogated and/or updated.
336 0410 1
337 0411 1     ATTR_CODE.rb.v Video rendition attribute code.
338 0412 1                 Bit 0   Bold
339 0413 1                 Bit 1   Reverse video
340 0414 1                 Bit 2   Blinking
341 0415 1                 Bit 3   Underscored
342 0416 1
343 0417 1     TEXT_LEN.rl.v Length of text string
344 0418 1
345 0419 1     TEXT_ADDR.rl.v Address of text string
346 0420 1
347 0421 1     CHAR_SET.rl.v Character set to use.
348 0422 1                 SMGSC_UNITED_KINGDOM
349 0423 1                 SMGSC_ASCII

```



```

350 0424 1 SMGSC_SPEC_GRAPHICS
351 0425 1 SMGSC_ALT_CHAR
352 0426 1 SMGSC_ALT_GRAPHICS
353 0427 1
354 0428 1 OVERFLOW.wl.r Optional. Address of longword in which
355 0429 1 to return the number of characters that
356 0430 1 did not fit on the line.
357 0431 1
358 0432 1 IMPLICIT INPUTS:
359 0433 1
360 0434 1 NONE
361 0435 1
362 0436 1 IMPLICIT OUTPUTS:
363 0437 1
364 0438 1 NONE
365 0439 1
366 0440 1 COMPLETION STATUS:
367 0441 1
368 0442 1 SSS_NORMAL Normal successful completion
369 0443 1
370 0444 1 SIDE EFFECTS:
371 0445 1
372 0446 1 NONE
373 0447 1
374 0448 1
375 0449 2 BEGIN
376 0450 2
377 0451 2 BUILTIN
378 0452 2 SCANC,
379 0453 2 NULLPARAMETER;
380 0454 2
381 0455 2 LOCAL
382 0456 2 TEXT_BUF : REF VECTOR [ ,BYTE], ! Addr of text buffer
383 0457 2 ATTR_BUF : REF VECTOR [ ,BYTE], ! Addr of attr buffer
384 0458 2 CHAR_BUF : REF VECTOR [ ,BYTE], ! Addr of char set buffer
385 0459 2 STATUS, ! status of subroutine calls
386 0460 2 WORK_OVERFLOW : INITIAL (0), ! no. of overflow chars
387 0461 2 BYTES_REMAINING, ! No. of bytes in input string yet to be
388 0462 2 processed.
389 0463 2 IN_POINTER; ! Current pointer into input string
390 0464 2
391 0465 2 LITERAL
392 0466 2 K_OVERFLOW_ARG = 6;
393 0467 2
394 0468 2 TEXT_BUF = .DCB [DCB_A_TEXT_BUF];
395 0469 2 ATTR_BUF = .DCB [DCB_A_ATTR_BUF];
396 0470 2 CHAR_BUF = .DCB [DCB_A_CHAR_SET_BUF];
397 0471 2
398 0472 2 BYTES_REMAINING = .TEXT_LEN;
399 0473 2 IN_POINTER = .TEXT_ADDR;
400 0474 2
401 0475 2 WHILE .BYTES_REMAINING NEQ 0
402 0476 2 DO
403 0477 2 BEGIN ! Overall loop
404 0478 2 LOCAL
405 0479 2 CHARS_TO_MOVE, ! No. of characters to move on this
406 0480 2 iteration

```

```

407 0481 3
408 0482
409 0483
410 0484
411 0485
412 0486
413 0487
414 0488
415 0489
416 0490
417 0491
418 0492
419 0493
420 0494
421 0495
422 0496
423 0497
424 0498
425 0499
426 0500
427 0501
428 0502
429 0503
430 0504
431 0505
432 0506
433 0507
434 0508
435 0509
436 0510
437 0511
438 0512
439 0513
440 0514
441 0515
442 0516
443 0517
444 0518
445 0519
446 0520
447 0521
448 0522
449 0523
450 0524
451 0525
452 0526
453 0527
454 0528
455 0529
456 0530
457 0531
458 0532
459 0533
460 0534
461 0535
462 0536
463 0537
  
```

```

PLACE_TO_MOVE,      ! Place to move from on this iteration
NEW_BYTES_REMAINING, ! No. of bytes remaining as returned
                    ! by SCANC
ADDR_DIFF;         ! Addr of char in input stream whose
                    ! index into scanc table yields
                    ! non-zero code.

!+
! See if any of the remaining input characters require special
! treatment.
-
SCANC ( BYTES_REMAINING,      ! No. of bytes remaining
        .IN_POINTER,        ! Current pointer to source
        CHAR_TABLE,        ! Address of SCANC table
        ALLOES;            ! Mask for ANDing
        NEW_BYTES_REMAINING, ! New remaining no. of bytes
                                ! including the byte which
                                ! caused the instruction to
                                ! halt. Is zero only if all
                                ! bytes did not satisfy search.
                                ! Addr of char in input stream
                                ! whose index into scanc table
                                ! yields non-zero code.

        ADDR_DIFF);

CHARS_TO_MOVE = .BYTES_REMAINING - .NEW_BYTES_REMAINING;
PLACE_TO_MOVE = .IN_POINTER;
IN_POINTER = .IN_POINTER + .CHARS_TO_MOVE;
BYTES_REMAINING = .NEW_BYTES_REMAINING;

!+
! Copy the appropriate number of characters into the text buffer
! and the appropriate number of copies of the attribute code
! into the attribute buffer.
-
IF .CHARS_TO_MOVE NEQ 0
THEN
  BEGIN
  LOCAL
  INDEX, ! 0-based index into BUFFER and ATTR_BUFFER.
  REMAINING_COLS;

  INDEX = $$MSG$LINEAR ( .DCB [DCB_W_CURSOR_ROW], .DCB [DCB_W_CURSOR_COL]);
  REMAINING_COLS = .DCB [DCB_W_NO_COLS] - .DCB [DCB_W_CURSOR_COL] + 1;
  IF .CHARS_TO_MOVE GTR .REMAINING_COLS
  THEN
    ! chars will overflow line
    BEGIN
    WORK_OVERFLOW = .BYTES_REMAINING +
                   (.CHARS_TO_MOVE - .REMAINING_COLS);
    CHARS_TO_MOVE = .REMAINING_COLS;
    END;

!+
! Move text into buffer.
-
CHSMOVE (.CHARS_TO_MOVE,      ! No. of chars
        .PLACE_TO_MOVE,     ! From
        TEXT_BUF [ .INDEX ] ); ! To
  
```



```
464 0538 4  
465 0539 4  
466 0540 4  
467 0541 4  
468 0542 4  
469 0543 4  
470 0544 4  
471 0545 5  
472 0546 5  
473 0547 5  
474 0548 5  
475 0549 5  
476 0550 5  
477 0551 5  
478 0552 5  
479 0553 5  
480 0554 5  
481 0555 4  
482 0556 4  
483 0557 4  
484 0558 4  
485 0559 4  
486 0560 4  
487 0561 4  
488 0562 4  
489 0563 4  
490 0564 4  
491 0565 4  
492 0566 4  
493 0567 4  
494 0568 4  
495 0569 4  
496 0570 4  
497 0571 4  
498 0572 4  
499 0573 4  
500 0574 4  
501 0575 4  
502 0576 4  
503 0577 4  
504 0578 4  
505 0579 4  
506 0580 4  
507 0581 4  
508 0582 4  
509 0583 5  
510 0584 5  
511 0585 5  
512 0586 5  
513 0587 5  
514 0588 5  
515 0589 5  
516 0590 5  
517 0591 5  
518 0592 5  
519 0593 5  
520 0594 5
```

```
!+  
! Rewrite attribute bytes. Normally the attributes are  
! passed to us, but for the 'autobended' case where escape  
! sequences are used, we should look at the default attributes  
! which may have been altered by SMG$SIM_TERM.  
-  
BEGIN  
LOCAL  
  WORK_ATTR;  
WORK_ATTR = .ATTR_CODE;  
IF .DCB [DCB_V_ALLOW_ESC]  
THEN  
  WORK_ATTR = .DCB [DCB_B_DEF_VIDEO_ATTR];  
CHSFILL (.WORK_ATTR, ! Char. to replicate  
  .CHARS_TO_MOVE, ! No. of times  
  ATTR_BUF [ .INDEX ] ); ! Destination  
END;  
!+  
! Write the character set bytes, if necessary.  
-  
IF .CHAR_BUF EQL 0 AND  
  .CHAR_SET NEQ SMG$C_ASCII  
THEN  
  0; ! first char set - alloc buffer  
IF .CHAR_BUF NEQ 0  
THEN  
  CHSFILL (.CHAR_SET,  
    .CHARS_TO_MOVE,  
    CHAR_BUF [.INDEX]);  
!+  
! Adjust resulting cursor position. Check for overflow.  
-  
DCB [DCB_W_CURSOR_COL] = .DCB [DCB_W_CURSOR_COL] +  
  CHARS_TO_MOVE;  
IF .DCB [DCB_W_CURSOR_COL] GTR .DCB [DCB_W_NO_COLS]  
THEN  
  DCB [DCB_W_CURSOR_COL] = .DCB [DCB_W_NO_COLS];  
IF .WORK_OVERFLOW NEQ 0  
THEN  
  EXITLOOP;  
END;  
IF .NEW_BYTES_REMAINING EQL 0  
THEN  
  EXITLOOP; ! Break out of loop -- we're done  
!+  
! Dispatch on the non-zero code located to see what special  
! action is needed.  
-  
CASE .CHAR_TABLE [.(.ADDR_DIFF) <0,8>] FROM 1 TO 10 OF
```



```

521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
001 PLL1013
554-1
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
  
```

```

0595
0596
0597
0598
0599
0600
0601
0602
0603
0604
0605
0606
0607
0608
0609
0610
0611
0612
0613
0614
0615
0616
0617
0618
0619
0620
0621
0622
0623
0624
0625
0626
0627
0628
0629
0630
0631
0632
0633
0634
0635
0636
0637
0638
0639
0640
0641
0642
0643
0644
0645
0646
0647
0648
0649
0650
0651
  
```

SET

[1]:

Hex Character Codes	ASCII Character
00 to 06	NUL to ACK
10 to 1A	DLE to SUB
1C to 1F	FS to US

Character can be discarded. Cursor is not advanced.

Special case if the user_graphic bit is set. That indicates a device-independent code which should be placed in the buffer for later interpretation by output. Notice that we are guaranteed that TEXT_ADDR contains only 1 character since only we call this routine.

IF (.ATTR_CODE AND ATTR_M_USER_GRAPHIC) NEQ 0
 THEN
 \$INSERT_CTRL_CHAR (.TEXT_ADDR);

[2]:

Hex Character Codes	ASCII Character
07	BEL

Character can be discarded. Cursor is not modified, and we call a routine to ring the bell now. (Note that if we had stored the bell in the attribute buffer, the bell would've been rung every time the screen was repainted.)

SMG\$RING_BELL (DCB [DCB_L_DID]);

[3]:

Hex Character Codes	ASCII Character
08	BS

Character can be discarded, but cursor must be backed up one column. Be careful about cursor already being in column 1.

BEGIN
 IF .DCB [DCB_W_CURSOR_COL] NEQ 1
 THEN
 DCB [DCB_W_CURSOR_COL] = .DCB [DCB_W_CURSOR_COL] -1;

END;

[4]:

Hex Character Codes	ASCII Character
09	HT

```

578 0652 3
579 0653 3
580 0654 3
581 0655 3
582 0656 3
583 0657 3
584 0658 3
585 0659 3
586 0660 3
587 0661 3
588 0662 4
589 0663 4
590 0664 4
591 0665 4
592 0666 4
593 0667 4
594 0668 4
595 0669 5
596 0670 5
597 0671 5
598 0672 5
599 0673 5
600 0674 5
601 0675 5
602 0676 4
603 0677 4
604 0678 4
605 0679 4
606 0680 4
607 0681 4
608 0682 4
609 0683 4
610 0684 4
611 0685 4
612 0686 4
613 0687 4
614 0688 4
615 0689 4
616 0690 4
617 0691 4
618 0692 4
619 0693 4
620 0694 4
621 0695 4
622 0696 4
623 0697 5
624 0698 5
625 0699 5
626 0700 5
627 0701 5
628 0702 5
629 0703 5
630 0704 5
631 0705 6
632 0706 5
633 0707 5
634 0708 5

```

Character can be discarded, but cursor must be advanced to next TAB stop and intervening character positions in the buffer must be left undisturbed.

TAB stops are assumed to be set in the following columns:
9, 17, 25, 33, 41, 49, 57, 65, 73 (width=80)

9, 17, 25, 33, 41, 49, 57, 65, 73, 81, 89, 97, 105, 113, 121, 129 (width=132)

```

BEGIN
+
Be careful about tabbing off the end of the line or beyond the end of the virtual display line.
-
IF NOT .DCB [DCB_V_DISPLAY_CONTROLS]
THEN
BEGIN
DCB [DCB_W_CURSOR_COL] =
      ((.DCB [DCB_W_CURSOR_COL]-1)/8+1)*8+1;
IF .DCB [DCB_W_CURSOR_COL] GTR .DCB [DCB_W_NO_COLS]
THEN
DCB [DCB_W_CURSOR_COL] = .DCB [DCB_W_NO_COLS];
END
ELSE
$INSERT_CTRL_CHAR (TAB);
END;

```

[5,6]:

Hex Character Codes	ASCII Character
0A	LF
0B	VT

Character can be discarded. Cursor must be advanced by one line. Don't advance beyond last line of display.

```

BEGIN
+
If cursor not at bottom, advance DCB [DCB_W_CURSOR_ROW] by one.
-
IF NOT .DCB [DCB_V_DISPLAY_CONTROLS]
THEN
BEGIN
IF .DCB [DCB_W_CURSOR_ROW] + 1 LEQ .DCB [DCB_W_BOTTOM_OF_SCRREG]
THEN
DCB [DCB_W_CURSOR_ROW] = .DCB [DCB_W_CURSOR_ROW] + 1
ELSE
SMG$$SCROLL_AREA (.DCB,
      .DCB [DCB_W_TOP_OF_SCRREG],
      .DCB [DCB_W_COL_START],
      (.DCB [DCB_W_BOTTOM_OF_SCRREG] -
      .DCB [DCB_W_TOP_OF_SCRREG] + 1),
      .DCB [DCB_W_NO_COLS],
      SMGSM_UP,

```



```

635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691

```

```

0709 5
0710 5
0711 5
0712 5
0713 5
0714 5
0715 5
0716 5
0717 5
0718 5
0719 5
0720 5
0721 5
0722 5
0723 5
0724 5
0725 5
0726 5
0727 5
0728 5
0729 4
0730 4
0731 4
0732 5
0733 5
0734 5
0735 5
0736 5
0737 5
0738 5
0739 5
0740 6
0741 5
0742 5
0743 5
0744 5
0745 5
0746 4
0747 4
0748 5
0749 5
0750 5
0751 5
0752 5
0753 5
0754 5
0755 5
0756 5
0757 5
0758 5
0759 4
0760 4
0761 4
0762 4
0763 4
0764 4
0765 3

```

```

1);
END
ELSE
BEGIN
LOCAL
CHAR;
CHAR = (.ADDR DIFF)<0,8>;
$INSERT_CTRL_CHAR (.CHAR);
END;
END;

[7]:
+
Hex Character Codes      ASCII Character
-----
OC                        FF
-
Character can be discarded. Effect is to clear the buffer
and reset the cursor to line 1 column 1.
BEGIN
IF NOT .DCB [DCB_V_DISPLAY_CONTROLS]
THEN
BEGIN
IF .DCB [DCB_W_CURSOR_ROW] + 1 LEQ .DCB [DCB_W_BOTTOM_OF_SCRREG]
THEN
DCB [DCB_W_CURSOR_ROW] = .DCB [DCB_W_CURSOR_ROW] + 1
ELSE
SMG$$SCROLL_AREA (.DCB,
.DCB [DCB_W_TOP_OF_SCRREG],
.DCB [DCB_W_COL_START],
(.DCB [DCB_W_BOTTOM_OF_SCRREG] -
.DCB [DCB_W_TOP_OF_SCRREG] + 1),
.DCB [DCB_W_NO_COLS],
SMG$M_UP,
1);
END
ELSE
$INSERT_CTRL_CHAR (FF);
END;

[8]:
+
Hex Character Codes      ASCII Character
-----
OD                        CR
-
Character can be discarded. Effect is to set cursor to
column 1 of current line.
BEGIN
IF NOT .DCB [DCB_V_DISPLAY_CONTROLS]
THEN
DCB [DCB_W_CURSOR_COL] = 1
ELSE
$INSERT_CTRL_CHAR (CR);
END;

```



```

692 0766
693 0767
694 0768
695 0769
696 0770
697 0771
698 0772
699 0773
700 0774
701 0775
702 0776
703 0777
704 0778
705 0779
706 0780
707 0781
708 0782
709 0783
710 0784
711 0785
712 0786
713 0787
714 0788
715 0789
716 0790
717 0791
718 0792
719 0793
720 0794
721 0795
722 0796
723 0797
724 0798
725 0799
726 0800
727 0801
728 0802
729 0803
730 0804
731 0805
732 0806
733 0807
734 0808
735 0809
736 0810
737 0811
738 0812
739 0813
740 0814
741 0815
742 0816
743 0817
744 0818
745 0819
746 0820
747 0821
748 0822
  
```

```

[9]:
+
Hex Character Codes      ASCII Character
-----
1B                        ESC
OE                        SO
OF                        SI

Character can be discarded. Subsequent characters need
to be inspected to see if they constitute a recognized
escape sequence whose effect must be simulated-- E.g.,
cursor setting, rendition setting.

SMG$$SIM_TERM processes the escape sequence, then returns
here to allow any remaining characters to be processed.
-
BEGIN
IF NOT .DCB [DCB_V_ALLOW_ESC]
THEN
RETURN (SMG$_STRTERESC) ! error from true SMG$
ELSE
BEGIN ! autobended - attempt to interpret
LOCAL
LEN_OF_SEQUENCE,
STATUS;
STATUS = SMG$$SIM_TERM (.DCB,
.BYTES_REMAINING,
.IN_POINTER, ! pass ptr to esc char
.LEN_OF_SEQUENCE);
IF NOT .STATUS THEN RETURN (.STATUS);

+
Update the number of bytes processed. Since there is
an automatic update (by 1 character) at the end of this
loop, don't count the ESC now.
-
BYTES_REMAINING = .BYTES_REMAINING - .LEN_OF_SEQUENCE + 1;
IN_POINTER = .IN_POINTER + .LEN_OF_SEQUENCE - 1;
END; ! autobended - attempt to interpret

END;

[10]:
+
Hex Character Codes      ASCII Character
-----
7F                        DEL

Character can be discarded.

! no special action

[INRANGE, OVRANGE]:
+
Should never get here -- there are no other codes in
CHAR_TABLE. If we do, we've got a problem.
  
```


006C
0186

09
0061
0118

	08	BE		06	A9	B0	000B9	
				18	AE	D5	000BB	6\$:
					03	13	000C0	
					022A	31	000C3	7\$:
				10	AE	D5	000C5	8\$:
					F8	13	000C8	
		52		0C	BE	9A	000CB	
		01	00000000	EF	42	8F	000CD	
		0055		001C			000D1	9\$:
		00DA		00DA			000DA	
		0211		01E0			000E2	
							000EA	

BLEQU	6\$	
MOVW	6(R9), @8(SP)	
TSTL	WORK_OVERFLOW	
BEQL	8\$	
BRW	41\$	
TSTL	NEW_BYTES_REMAINING	
BEQL	7\$	
MOVZBL	@ADDR_DIFF, R2	
CASEB	CHAR_TABLE[R2], #1, #9	
.WORD	10\$-9\$,-	
	12\$-9\$,-	
	13\$-9\$,-	
	15\$-9\$,-	
	21\$-9\$,-	
	21\$-9\$,-	
	23\$-9\$,-	
	31\$-9\$,-	
	38\$-9\$,-	
	40\$-9\$	

0578
0580

0586

0594

		50	00000000G	8F	D0	000EE				
					04	000F5				
	49	08		06	A9	3C	000F6	10\$:		
				06	A9	3C	000FB			
				28	A9	3C	000FF			
					50	C2	00103			
				28	A9	3C	00106			
					50	D7	0010A			
				06	A9	3C	0010C			
					51	C4	00110			
				2A	A9	9E	00113			
					62	3C	00117			
				FF	A140	9E	0011A			
					53	D5	0011F			
					03	14	00121			
				016C		31	00123			
				00	EF	00126		11\$:		
				00BA		31	0012C			
				38	A9	9F	0012F	12\$:		
			00000000G	00	01	FB	00132			
					77	11	00139			
				01	2A	A9	B1	0013B	13\$:	
					71	13	0013F			
				2A	A9	B7	00141			
					6C	11	00144	14\$:		
				2A	A9	9E	00146	15\$:		
	18	2F			02	E0	0014A			
					63	3C	0014F			
					50	D7	00152			
					50	08	C6	00154		
					03	78	00157			
	51				09	A1	0015B			
	63				63	B1	0015F			
		06			4D	1B	00163			
					47	11	00165			
					52	06	A9	3C	00167	16\$:
					50	28	A9	3C	0016B	
					52	50	C2	0016F		

MOVL	#SMGS_FATERRLIB, R0	
RET		
BBC	#6, ATTR_CODE, 14\$	
MOVZWL	6(R9), REMAINING_COLS	
MOVZWL	40(R9), R0	
SUBL2	R0, REMAINING_COLS	
MOVZWL	40(R9), R0	
DECL	R0	
MOVZWL	6(R9), R1	
MULL2	R1, R0	
MOVAB	42(R9), R2	
MOVZWL	(R2), R1	
MOVAB	-1(R1)[R0], INDEX	
TSTL	REMAINING_COLS	
BGTR	11\$	
BRW	34\$	
EXTZV	#0, #4, TEXT_ADDR, R3	
BRW	22\$	
PUSHAB	56(R9)	
CALLS	#1, SMG\$RING_BELL	
BRB	20\$	
CMPW	42(R9), #1	
BEQL	20\$	
DECW	42(R9)	
BRB	20\$	
MOVAB	42(R9), R3	
BBS	#2, 47(R9), 16\$	
MOVZWL	(R3), R0	
DECL	R0	
DIVL2	#8, R0	
ASHL	#3, R0, R1	
ADDW3	#9, R1, (R3)	
CMPW	(R3), 6(R9)	
BLEQU	20\$	
BRB	19\$	
MOVZWL	6(R9), REMAINING_COLS	
MOVZWL	40(R9), R0	
SUBL2	R0, REMAINING_COLS	

0825

0612

0614

0627

0640

0642

0594

067^

0667

0671

0672

0674

0677

			50	28	A9	3C	00172		MOVZWL	40(R9), R0		
					50	D7	00176		DECL	R0		
			51	06	A9	3C	00178		MOVZWL	6(R9), R1		
			50		51	C4	0017C		MULL2	R1, R0		
			51		63	3C	0017F		MOVZWL	(R3), R1		
			51	FF	A140	9E	00182		MOVAB	-1(R1)[R0], INDEX		
					52	D5	00187		TSTL	REMAINING_COLS		
					06	14	00189		BGTR	17\$		
	18	AE			57	D0	0018B		MOVL	BYTES_REMAINING, WORK_OVERFLOW		
					15	11	0018F		BRB	18\$		
			50	90	8F	90	00191	17\$:	MOVB	#-112, SHIFT_NIBBLE		
			614B		50	90	00195		MOVB	SHIFT_NIBBLE, (INDEX)[TEXT_BUF]		
			50	08	AC	9A	00199		MOVZBL	ATTR_CODE, WORK_ATTR		
50		01	06		01	F0	0019D		INSV	#1, #6, #1, WORK_ATTR		
			614A		50	90	001A2		MOVB	WORK_ATTR, (INDEX)[ATTR_BUF]		
					63	B6	001A6	18\$:	INCW	(R3)		
	06	A9			63	B1	001A8		CMPW	(R3), 6(R9)		
					5A	12	001AC		BNEQ	25\$		
			63	06	A9	B0	001AE	19\$:	MOVW	6(R9), (R3)		
					7D	11	001B2	20\$:	BRB	27\$		
					02	E1	001B4	21\$:	BBC	#2, 47(R9), 24\$		
	3E	2F	A9		52	D0	001B9		MOVL	R2, CHAR		
			54		06	A9	001BC		MOVZWL	6(R9), REMAINING_COLS		
			53		28	A9	001C0		MOVZWL	40(R9), R0		
			50		50	C2	001C4		SUBL2	R0, REMAINING_COLS		
			53		28	A9	001C7		MOVZWL	40(R9), R0		
			50		50	D7	001CB		DECL	R0		
			51	06	A9	3C	001CD		MOVZWL	6(R9), R1		
			50		51	C4	001D1		MULL2	R1, R0		
			52	2A	A9	9E	001D4		MOVAB	42(R9), R2		
			51		62	3C	001D8		MOVZWL	(R2), R1		
			51	FF	A140	9E	001DB		MOVAB	-1(R1)[R0], INDEX		
					53	D5	001E0		TSTL	REMAINING_COLS		
					75	15	001E2		BLEQ	29\$		
			53		00	EF	001E4		EXTZV	#0, #4, CHAR, R3		
53		54	04		04	78	001E9	22\$:	ASHL	#4, R3, R3		
		53	53		53	90	001ED		MOVB	R3, SHIFT_NIBBLE		
			50		6C	11	001F0		BRB	30\$		
					02	E0	001F2	23\$:	BBS	#2, 47(R9), 28\$		
		3C	2F	A9	28	A9	001F7	24\$:	MOVZWL	40(R9), R0		
			50		50	D6	001FB		INCL	R0		
					00	ED	001FD		CMPZV	#0, #16, 74(R9), R0		
50		4A	A9		05	19	00203		BLSS	26\$		
					28	A9	00205		INCW	40(R9)		
					62	11	00208	25\$:	BRB	32\$		
					01	DD	0020A	26\$:	PUSHL	#1		
					01	DD	0020C		PUSHL	#1		
			7E	06	A9	3C	0020E		MOVZWL	6(R9), -(SP)		
			50	4A	A9	3C	00212		MOVZWL	74(R9), R0		
			51	48	A9	3C	00216		MOVZWL	72(R9), R1		
			50		51	C2	0021A		SUBL2	R1, R0		
					01	A0	9F	0021D	PUSHAB	1(R0)		
			7E	04	A9	3C	00220		MOVZWL	4(R9), -(SP)		
			7E	48	A9	3C	00224		MOVZWL	72(R9), -(SP)		
					59	DD	00228		PUSHL	R9		
			00000000G	00	07	FB	0022A		CALLS	#7, SMG\$\$\$SCROLL_AREA		
					39	11	00231	27\$:	BRB	32\$		

.....
 0594
 0695
 0715
 0716

 0730
 0733

 0735

 0737

 0742
 0741

 0740
 0739
 0738
 0737

 0730

				53	06	A9	3C	00233	28\$:	MOVZWL	6(R9), REMAINING_COLS		0747
				50	28	A9	3C	00237		MOVZWL	40(R9), R0		
				53		50	C2	0023B		SUBL2	R0, REMAINING_COLS		
				50	28	A9	3C	0023E		MOVZWL	40(R9), R0		
						50	D7	00242		DECL	R0		
				51	06	A9	3C	00244		MOVZWL	6(R9), R1		
				50		51	C4	00248		MULL2	R1, R0		
				52	2A	A9	9E	0024B		MOVAB	42(R9), R2		
				51		62	3C	0024F		MOVZWL	(R2), R1		
				51	FF	A140	9E	00252		MOVAB	-1(R1)[R0], INDEX		
						53	D5	00257		TSTL	REMAINING_COLS		
						37	15	00259	29\$:	BLEQ	34\$		
				50		3F	92	0025B		MCOMB	#63, SHIFT_NIBBLE		
						3B	11	0025E	30\$:	BRB	36\$		
	05			52	2A	A9	9E	00260	31\$:	MOVAB	42(R9), R2		0762
		2F		A9		02	E0	00264		BBS	#2, 47(R9), 33\$		0760
				62		01	B0	00269		MOVW	#1, (R2)		0762
						7D	11	0026C	32\$:	BRB	40\$		
				53	06	A9	3C	0026E	33\$:	MOVZWL	6(R9), REMAINING_COLS		0764
				50	28	A9	3C	00272		MOVZWL	40(R9), R0		
				53		50	C2	00276		SUBL2	R0, REMAINING_COLS		
				50	28	A9	3C	00279		MOVZWL	40(R9), R0		
						50	D7	0027D		DECL	R0		
				51	06	A9	3C	0027F		MOVZWL	6(R9), R1		
				50		51	C4	00283		MULL2	R1, R0		
				51		62	3C	00286		MOVZWL	(R2), R1		
				51	FF	A140	9E	00289		MOVAB	-1(R1)[R0], INDEX		
						53	D5	0028E		TSTL	REMAINING_COLS		
						06	14	00290		BGTR	35\$		
		18		AE		57	D0	00292	34\$:	MOVL	BYTES_REMAINING, WORK_OVERFLOW		
						14	11	00296		BRB	37\$		
				50		30	8E	00298	35\$:	MNEGB	#48, SHIFT_NIBBLE		
			614B			50	90	0029B	36\$:	MOVW	SHIFT_NIBBLE, (INDEX)[TEXT_BUF]		
			50		08	AC	9A	0029F		MOVZBL	ATTR_CODE, WORK_ATTR		
			06			01	F0	002A3		INSV	#1, #6, #1, WORK_ATTR		
50		01		614A		50	90	002AB		MOVW	WORK_ATTR, (INDEX)[ATTR_BUF]		
						62	B6	002AC	37\$:	INCW	(R2)		
				06	A9	62	B1	002AE		CMPW	(R2), 6(R9)		
						37	12	002B2		BNEQ	40\$		
				62	06	A9	B0	002B4		MOVW	6(R9), (R2)		
						31	11	002B8		BRB	40\$		0594
		08		34	A9	05	E0	002BA	38\$:	BBS	#5, 52(R9), 39\$		0784
				50	00000000G	8F	D0	002BF		MOVL	#SMG\$_STRTERESC, R0		0786
							04	002C6		RET			
						1C	AE	002C7	39\$:	PUSHAB	LEN OF SEQUENCE		0792
				7E		57	7D	002CA		MOVQ	BYTES_REMAINING, -(SP)		0793
						59	DD	002CD		PUSHL	R9		0792
			00000000G	00		04	FB	002CF		CALLS	#4, SMG\$SIM_TERM		
				36		50	E9	002D6		BLBC	STATUS, 44\$		0796
				50		57	1C	AE	C3	002D9	LEN OF SEQUENCE, BYTES_REMAINING, R0		0803
				57	01	A0	9E	002DE		MOVAB	1(R0), -BYTES_REMAINING		
				50		58	1C	AE	C1	002E2	LEN OF SEQUENCE, IN_POINTER, R0		0804
				58	FF	A0	9E	002E7		MOVAB	-1(R0), -IN_POINTER		
						58	D6	002EB	40\$:	INCL	IN_POINTER		0834
						57	D7	002ED		DECL	BYTES_REMAINING		0835
						FD2B	31	002EF		BRW	1\$		0475
				06	A9	2A	A9	B1	002F2	41\$:	CMPW	42(R9), 6(R9)	0838

SMG\$SPUT_TEXT_T Put text to display buffer
1-013 SMG\$SPUT_TEXT_TO_BUFFER - Put text to buffer

M 14
9-Jan-1985 22:03:37
2-Oct-1984 12:58:20

VAX-11 Bliss-32 V4.0-742
[SMGRTL.BUGSRC]SMGPUTTEX.B32;1

Page 21
(5)

34	A9		04	12	002F7		BNEQ	42\$		
	06		02	88	002F9		BISB2	#2, 52(R9)		0840
			6C	91	002FD	42\$:	CMPB	(AP), #6		0842
			0A	1F	00300		BLSSU	43\$		
		18	AC	D5	00302		TSTL	24(AP)		
			05	13	00305		BEQL	43\$		
18	BC	18	AE	D0	00307		MOVL	WORK OVERFLOW, @OVERFLOW		0844
	50		01	D0	0030C	43\$:	MOVL	#1, R0		0846
			04	0030F	44\$:		RET			0847

; Routine Size: 784 bytes, Routine Base: _SMG\$CODE + 0000

; 774 0848 1 !<BLF/PAGE>

```

SMG$$PUT_TEXT_T Put text to display buffer          N 14
1-013           SMG$$PUT_TEXT_TO_BUFFER - Put text to buffer 9-Jan-1985 22:03:37
                                                    2-Oct-1984 12:58:20
                                                    VAX-11 Bliss-32 V4.0-742
                                                    [SMGRTL.BUGSRC]SMGPUTTEX.B32;1
                                                    Page 22
                                                    (6)
: 776           0849 1 END
: 777           0850 1
: 778           0851 0 ELUDOM
! End of module SMG$$PUT_TEXT_TO_BUFFER

```

PSECT SUMMARY

Name	Bytes	Attributes
_SMG\$DATA	260	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, PIC, ALIGN(2)
_SMG\$CODE	784	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(2)

Library Statistics

File	Total	Symbols Loaded	Percent	Pages Mapped	Processing Time
_\$255\$DIIA18:[SYSLIB]STARLET.L32;1	9776	5	0	581	00:01.1
_\$255\$DUJA18:[SMGRTL.OBJ]RTLLIB.L32;1	36	0	0	8	00:00.1
_\$255\$DUA18:[SMGRTL.OBJ]SMGLIB.L32;1	469	19	4	38	00:00.4

COMMAND QUALIFIERS

```

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/NOTRACE/LIS=LIS$:SMGPUTTEX/OBJ=OBJ$:SMGPUTTEX MSRC$:SMGPUTTEX/UPDATE=(BUG$:SMGPUTTEX
)

```

```

: Size:          784 code + 260 data bytes
: Run Time:      00:24.2
: Elapsed Time: 00:28.9
: Lines/CPU Min: 2110
: Lexemes/CPU-Min: 17915
: Memory Used: 354 pages
: Compilation Complete

```


