


```

LL          IIIIII  BBBB8888  TTTTTTTTTT  PPPPPPPP  AAAAAA  RRRRRRRR  SSSSSSSS  FEEEEEEEEE
LL          IIIIII  BBBB8888  TTTTTTTTTT  PPPPPPPP  AAAAAA  RRRRRRRR  SSSSSSSS  FEEEEEEEEE
LL          II      88      88      TT      PP      PP  AA      AA  RR      RR  SS      FEEEEEEEEE
LL          II      88      88      TT      PP      PP  AA      AA  RR      RR  SS      FEEEEEEEEE
LL          II      88      88      TT      PP      PP  AA      AA  RR      RR  SS      FEEEEEEEEE
LL          II      88      88      TT      PP      PP  AA      AA  RR      RR  SS      FEEEEEEEEE
LL          II      BBBB8888  TT      PPPPPPPP  AA      AA  RRRRRRRR  SSSSSS  FEEEEEEEEE
LL          II      BBBB8888  TT      PPPPPPPP  AA      AA  RRRRRRRR  SSSSSS  FEEEEEEEEE
LL          II      88      88      TT      PP      AA      AA  RR      RR  SS      FEEEEEEEEE
LL          II      88      88      TT      PP      AA      AA  RR      RR  SS      FEEEEEEEEE
LL          II      88      88      TT      PP      AA      AA  RR      RR  SS      FEEEEEEEEE
LL          II      88      88      TT      PP      AA      AA  RR      RR  SS      FEEEEEEEEE
LLLLLLLLLLL IIIIII  BBBB8888  TT      PP      AA      AA  RR      RR  SSSSSSSS  FEEEEEEEEE
LLLLLLLLLLL IIIIII  BBBB8888  TT      PP      AA      AA  RR      RR  SSSSSSSS  FEEEEEEEEE

```

```

LL          IIIIII  SSSSSSSS
LL          IIIIII  SSSSSSSS
LL          II      SS
LL          II      SS
LL          II      SS
LL          II      SS
LL          II      SSSSSS
LL          II      SSSSSS
LL          II      SS
LL          II      SS
LL          II      SS
LL          II      SS
LLLLLLLLLLL IIIIII  SSSSSSSS
LLLLLLLLLLL IIIIII  SSSSSSSS

```



```

1 0001 0 MODULE LIBSTPARSE (
2 0002 0 IDENT = 'V04-000'
3 0003 0 ) =
4 0004 1 BEGIN
5 0005 1
6 0006 1
7 0007 1 *****
8 0008 1 *
9 0009 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
10 0010 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
11 0011 1 * ALL RIGHTS RESERVED.
12 0012 1 *
13 0013 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
14 0014 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
15 0015 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
16 0016 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
17 0017 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
18 0018 1 * TRANSFERRED.
19 0019 1 *
20 0020 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
21 0021 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
22 0022 1 * CORPORATION.
23 0023 1 *
24 0024 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
25 0025 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
26 0026 1 *
27 0027 1 *
28 0028 1 *****
29 0029 1
30 0030 1 ++
31 0031 1
32 0032 1 FACILITY: System Library
33 0033 1
34 0034 1 ABSTRACT:
35 0035 1
36 0036 1 TPARSE is a general purpose state table driven parser. Its
37 0037 1 general design is that of a finite state parser; however,
38 0038 1 some of its features allow non-deterministic parsing and
39 0039 1 limited use as a push-down parser. The input string is parsed
40 0040 1 by interpreting the transitions in the user supplied state
41 0041 1 table; user supplied action routines are called as indicated
42 0042 1 in the state table to provide the semantics associated with
43 0043 1 the table specified syntax.
44 0044 1
45 0045 1 ENVIRONMENT:
46 0046 1
47 0047 1 Native node star processor; no operating system facilities are
48 0048 1 needed. Minimum of 21 longwords of stack required.
49 0049 1
50 0050 1 --
51 0051 1
52 0052 1
53 0053 1 AUTHOR: Andrew C. Goldstein, CREATION DATE: 14-Oct-1976 13:55
54 0054 1
55 0055 1 MODIFIED BY:
56 0056 1
57 0057 1 V03-004 ACG0396 Andrew C. Goldstein, 31-Jan-1984 22:10

```

```

58      0058 1      Add multi-national characters to alphabetics; add a
59      0059 1      character data type for hex digits
60      0060 1
61      0061 1      V03-003 ACG0392      Andrew C. Goldstein,      19-Jan-1984  21:32
62      0062 1      Add match-all identifier to ident parsing; also add
63      0063 1      TPAS_FILESPEC item.
64      0064 1
65      0065 1      V03-002 ACG0351      Andrew C. Goldstein,      22-Aug-1983  15:42
66      0066 1      Add wild cards to UIC and IDENT parsing; disallow explicit
67      0067 1      -1 values for group and member
68      0068 1
69      0069 1      V03-001 ACG0345      Andrew C. Goldstein,      29-Jul-1983  15:20
70      0070 1      Add UIC and identifier token types
71      0071 1
72      0072 1      V02-009 ACG0263      Andrew C. Goldstein,      8-Feb-1982  23:35
73      0073 1      Fix token pointer after subexpression parse failure
74      0074 1
75      0075 1      V02-008 ACG0203      Andrew C. Goldstein,      28-Apr-1981  15:44
76      0076 1      Propagate return status past subexpression calls
77      0077 1
78      0078 1      V02-007 ACG0155      Andrew C. Goldstein,      12-Mar-1980  10:04
79      0079 1      Allow subexpressions to change the command string
80      0080 1
81      0081 1      1-006  ACG26405      Andrew C. Goldstein,      6-Dec-1979  20:14
82      0082 1      Change string count in control block to word
83      0083 1
84      0084 1      1-005  ACG0088      Andrew C. Goldstein,      4-Dec-1979  17:22
85      0085 1      Propagate action routine status through subexpression returns
86      0086 1
87      0087 1      V0004  ACG0019      Andrew C. Goldstein,      26-Mar-1978  16:27
88      0088 1      Add return of action routine status; fix trailing blanks problem
89      0089 1      Change PSECTS to new procedure library standards
90      0090 1
91      0091 1      **
92      0092 1
93      0093 1
94      0094 1      LIBRARY 'SYSS$LIBRARY:STARLET.L32';
95      0095 1      LIBRARY 'SYSS$LIBRARY:TPAMAC.L32';
96      0096 1
97      0097 1
98      0098 1
99      0099 1      PSECT
100     0100 1      CODE          = _LIB$CODE (SHARE, PIC),
101     0101 1      PLIT          = _LIB$CODE (SHARE, PIC);
102     0102 1
103     0103 1      LINKAGE
104     0104 1      L_GETSTRING    = CALL (STANDARD, STANDARD)
105     0105 1      : GLOBAL (CHAR_COUNT = 6);
106     0106 1
107     0107 1      FORWARD ROUTINE
108     0108 1      LIB$PARSE,      : main parser routine
109     0109 1      GETSTRING      : L_GETSTRING,    : extract a basic string token
110     0110 1      PARSE_FILESPEC, : parse filespec string
111     0111 1      PARSE_IDENT;   : parse identifier string
112     0112 1
113     0113 1
114     0114 1      EXTERNAL ROUTINE

```

LIB\$TPARSE
V04-000

C 11
16-Sep-1984 02:20:18
16-Sep-1984 13:35:03

VAX-11 BLISS-32 V4.0-742
[VMSLIB.SRC]TPARSE.B32;1

Page 3
(1)

LIB\$
V04-

:	115	0115	1	LIB\$CVT_DTB	:	ADDRESSING_MODE (GENERAL),	!	decimal to binary conversion
:	116	0116	1	LIB\$CVT_OTB	:	ADDRESSING_MODE (GENERAL),	!	octal to binary conversion
:	117	0117	1	LIB\$CVT_HTB	:	ADDRESSING_MODE (GENERAL),	!	hex to binary conversion

0000

0000

0000

```
119 0118 1  | Local macros and symbol definitions
120 0119 1  |
121 0120 1  |
122 0121 1  |
123 0122 1  | LITERAL
124 0123 1  |
125 0124 1  | Character codes
126 0125 1  |
127 0126 1  |     SPACE           = %0'40'      | space character
128 0127 1  |     TERMINATOR     = %0'377'    | keyword string terminator
129 0128 1  |     TAB             = %0'11'    | tab character
130 0129 1  |
131 0130 1  | String types (input to GETSTRING routine)
132 0131 1  |
133 0132 1  |     SPACES         = 0.
134 0133 1  |     NUMERIC        = 1.
135 0134 1  |     HEXNUMERIC     = 2.
136 0135 1  |     ALPHANUMERIC   = 3.
137 0136 1  |     SYMBOL         = 4.
138 0137 1  |
139 0138 1  | Token types
140 0139 1  |
141 0140 1  |     $FILESPEC     = %X'FF' AND TPAS_FILESPEC, | filespec expression
142 0141 1  |     $UIC          = %X'FF' AND TPAS_UIC,      | UIC expression
143 0142 1  |     $IDENT       = %X'FF' AND TPAS_IDENT,    | identifier expression
144 0143 1  |     $ANY         = %X'FF' AND TPAS_ANY,      | any single character
145 0144 1  |     $ALPHA       = %X'FF' AND TPAS_ALPHA,    | any alphabetic character
146 0145 1  |     $DIGIT       = %X'FF' AND TPAS_DIGIT,    | any numeric character
147 0146 1  |     $STRING      = %X'FF' AND TPAS_STRING,   | any alphanumeric string
148 0147 1  |     $SYMBOL      = %X'FF' AND TPAS_SYMBOL,   | any symbol constituent set string
149 0148 1  |     $BLANK       = %X'FF' AND TPAS_BLANK,    | any string of spaces and tabs
150 0149 1  |     $DECIMAL     = %X'FF' AND TPAS_DECIMAL,  | decimal number
151 0150 1  |     $OCTAL       = %X'FF' AND TPAS_OCTAL,    | octal number
152 0151 1  |     $HEX         = %X'FF' AND TPAS_HEX,      | hexadecimal number
153 0152 1  |     $LAMBDA      = %X'FF' AND TPAS_LAMBDA,   | empty string
154 0153 1  |     $EOS         = %X'FF' AND TPAS_EOS,      | end of string
155 0154 1  |     $SUBEXPR     = %X'FF' AND TPAS_SUBEXPR,  | subexpression
156 0155 1  |     NULL_MATCH   = TPAS_LAMBDA,             | codes geq match null strings
157 0156 1  |     HIGH_ASCII   = 255,                     | highest ASCII character code
158 0157 1  |     KEYWORD      = 256,                     | start of keyword codes
159 0158 1  |     HIGH_KEYWORD = 475,                     | highest keyword code
160 0159 1  |     LOW_SPECIAL  = $FILESPEC,               | first special type code
161 0160 1  |     HIGH_SPECIAL = $SUBEXPR;                | last special type code
162 0161 1  |
163 0162 1  | Macros to determine character types
164 0163 1  |
165 0164 1  |
166 0165 1  | MACRO
167 M 0166 1  |     IS_ALPHABETIC (CHAR) = (SELECTONE CHAR OF
168 M 0167 1  |     SET
169 M 0168 1  |     [%ASCII 'A' TO %ASCII 'Z',
170 M 0169 1  |     %ASCII 'a' TO %ASCII 'z',
171 M 0.70 1  |     %X'CO' TO %X'CF',
172 M 0171 1  |     %X'D1' TO %X'DD',
173 M 0172 1  |     %X'DF' TO %X'EF',
174 M 0173 1  |     %X'F1' TO %X'FD']): 1;
175 M 0174 1  |     [OTHERWISE]: 0;
```

```
176 M 0175 1 TES
177 0176 1 )%,
178 0177 1
179 M 0178 1 IS_NUMERIC (CHAR) = (SELECTONE CHAR OF
180 0179 1 SET
181 0180 1 [ASCII '0' TO ASCII '9']: 1;
182 0181 1 [OTHERWISE]: 0;
183 M 0182 1 TES
184 0183 1 )%,
185 0184 1
186 M 0185 1 IS_HEX (CHAR) = (SELECTONE CHAR OF
187 0186 1 SET
188 0187 1 [ASCII '0' TO ASCII '9',
189 0188 1 ASCII 'A' TO ASCII 'F',
190 0189 1 ASCII 'a' TO ASCII 'f']: 1;
191 0190 1 [OTHERWISE]: 0;
192 M 0191 1 TES
193 0192 1 )%,
194 0193 1
195 M 0194 1 IS_SYMBOL (CHAR) = (SELECTONE CHAR OF
196 0195 1 SET
197 0196 1 [ASCII ' ', ASCII '$']: 1;
198 0197 1 [OTHERWISE]: 0;
199 M 0198 1 TES
200 0199 1 )%,
201 0200 1
202 M 0201 1 IS_SPACE (CHAR) = (SELECTONE CHAR OF
203 0202 1 SET
204 0203 1 [SPACE, TAB]: 1;
205 0204 1 [OTHERWISE]: 0;
206 M 0205 1 TES
207 0206 1 )%,
208 0207 1
209 0208 1 ! Macros to fetch state table entries, for both data items and self
210 0209 1 ! relative addresses.
211 0210 1 !
212 0211 1
213 0212 1 MACRO
214 M 0213 1 GET_BYTE (POINTER) =
215 M 0214 1 (POINTER = .POINTER + 1; .(.POINTER-1)<0,8>)
216 0215 1 %,
217 0216 1
218 M 0217 1 GET_WORD (POINTER) =
219 M 0218 1 (POINTER = .POINTER + 2; .(.POINTER-2)<0,16>)
220 0219 1 %,
221 0220 1
222 M 0221 1 GET_LONG (POINTER) =
223 M 0222 1 (POINTER = .POINTER + 4; .(.POINTER-4)<0,32>)
224 0223 1 %,
225 0224 1
226 M 0225 1 REL_WORD (POINTER) =
227 M 0226 1 (POINTER = .POINTER + 2; .POINTER + .(.POINTER-2)<0,16,1>)
228 0227 1 %,
229 0228 1
230 M 0229 1 REL_LONG (POINTER) =
231 M 0230 1 (POINTER = .POINTER + 4; .POINTER + .(.POINTER-4)<0,32,1>)
232 0231 1 %;
```

```
.. 233      0232 1
.. 234      0233 1 ! Structure definition for the transition length table. It consists of 64
.. 235      0234 1 ! entries, indexed by the 6 transition option bits. Each entry is a 4 bit
.. 236      0235 1 ! nibble giving the number of words that are used by that transition
.. 237      0236 1 ! option mask.
.. 238      0237 1 !
.. 239      0238 1 !
.. 240      0239 1 STRUCTURE
.. 241      0240 1     NIBBLEVECTOR [I: N] =
.. 242      0241 1         [(N+1)/2]
.. 243      0242 1         NIBBLEVECTOR<I*4, 4, 0>;
.. 244      0243 1 !
.. 245      0244 1 ! Macro to generate nibbles in a PLIT expression.
.. 246      0245 1 !
.. 247      0246 1 !
.. 248      0247 1 MACRO
.. 249      0248 1     NIBBLE [A, B] =
.. 250      0249 1         BYTE (B*4 + A)
.. 251      0250 1         %;
.. 252      0251 1 !
.. 253      0252 1 ! The transition length table.
.. 254      0253 1 !
.. 255      0254 1 !
.. 256      0255 1 BIND
.. 257      0256 1     LENGTH_TABLE = UPLIT (NIBBLE (0,1,1,2,2,3,3,4,2,3,3,4,4,5,5,6,
.. 258      0257 1         2,3,3,4,4,5,5,6,4,5,5,6,6,7,7,8,
.. 259      0258 1         2,3,3,4,4,5,5,6,4,5,5,6,6,7,7,8,
.. 260      0259 1         4,5,5,6,6,7,7,8,6,7,7,8,8,9,9,10
.. 261      0260 1         ))
.. 262      0261 1     : NIBBLEVECTOR;
```



```
264 0262 1 GLOBAL ROUTINE LIB$TPARSE (STATE_VECTOR, START_STATE, KEYTAB) =
265 0263 1
266 0264 1 :++
267 0265 1
268 0266 1 FUNCTIONAL DESCRIPTION:
269 0267 1
270 0268 1     This routine is the main parser routine.
271 0269 1
272 0270 1
273 0271 1 CALLING SEQUENCE:
274 0272 1     LIB$TPARSE (ARG1, ARG2, ARG3)
275 0273 1
276 0274 1
277 0275 1 INPUT PARAMETERS:
278 0276 1     ARG1 = address of state vector, containing:
279 0277 1         options longword
280 0278 1             bit 0 set to match blanks and tabs
281 0279 1             clear to ignore blanks and tabs
282 0280 1             bit 1 set to allow minimum abbreviation of keywords
283 0281 1             clear to use match count
284 0282 1             high byte = keyword match count (0 = exact)
285 0283 1             string descriptor of string to be parsed
286 0284 1             data made available to action routines:
287 0285 1                 string descriptor of matching token
288 0286 1                 single character token
289 0287 1                 numerical value of numeric token
290 0288 1     ARG2 = address of starting state in state table
291 0289 1     ARG3 = address of keyword table
292 0290 1
293 0291 1 IMPLICIT INPUTS:
294 0292 1     NONE
295 0293 1
296 0294 1 OUTPUT PARAMETERS:
297 0295 1     string descriptor pointed to by ARG1 updated to indicate
298 0296 1     string not processed by the parser
299 0297 1
300 0298 1 IMPLICIT OUTPUTS:
301 0299 1     NONE
302 0300 1
303 0301 1 ROUTINE VALUE:
304 0302 1     1 if successful parse
305 0303 1     LIB$SYNTAXERR if unsuccessful parse
306 0304 1     LIB$INVTYPE if state table is invalid
307 0305 1     value of action routine rejecting transition
308 0306 1
309 0307 1 SIDE EFFECTS:
310 0308 1     none except as produced by user's action routines
311 0309 1
312 0310 1 --
313 0311 1
314 0312 1
315 0313 2 BEGIN
316 0314 2
317 0315 2 LOCAL
318 0316 2     STATE,                ! state table pointer
319 0317 2     LAST_SPACE,          ! character count of last string of spaces
320 0318 2     ACT_STATUS          : BLOCK [4, BYTE], ! status return of action routine
```

```

: 321      0319 2      TYPE          : BLOCK [4, BYTE]; ! syntax type of current transition
: 322      0320 2
: 323      0321 2
: 324      0322 2 MAP
: 325      0323 2      STATE_VECTOR : REF BLOCK [, BYTE], ! user state vector
: 326      0324 2      KEYTAB   : REF VECTOR[, WORD, SIGNED]; ! keyword table
: 327      0325 2
: 328      0326 2 BUILTIN
: 329      0327 2      TESTBITSC;
: 330      0328 2
: 331      0329 2
: 332      0330 2 ! TPARSE data area available to action routines.
: 333      0331 2
: 334      0332 2 MACRO
: 335      0333 2      STATE_LENGTH = STATE_VECTOR[TPASL_COUNT]%, ! length of user supplied state vector
: 336      0334 2      STRING_COUNT = (STATE_VECTOR[TPASL_STRINGCNT])<0,16>%, ! byte count of string being parsed
: 337      0335 2      STRING_POINTER = STATE_VECTOR[TPASL_STRINGPTR]%, ! address of string being parsed
: 338      0336 2      TOKEN_COUNT = STATE_VECTOR[TPASL_TOKENCNT]%, ! byte count of current token
: 339      0337 2      TOKEN_POINTER = STATE_VECTOR[TPASL_TOKENPTR]%, ! address of current token
: 340      0338 2      STATE_CHAR = STATE_VECTOR[TPASL_CHAR]%, ! current single character token
: 341      0339 2      STATE_NUMBER = STATE_VECTOR[TPASL_NUMBER]%, ! numerical value of number token
: 342      0340 2      STATE_PARAM = STATE_VECTOR[TPASL_PARAM]%, ! action routine parameter from state table
: 343      0341 2      MCOUNT = STATE_VECTOR[TPASB_MCOUNT]%, ! match abbreviation count
: 344      0342 2      SPACE_FLAG = STATE_VECTOR[TPASV_BLANKS]%, ! process spaces explicitly
: 345      0343 2      ABBRFM_FLAG = STATE_VECTOR[TPASV_ABBRFM]%, ! allow first match abbreviations
: 346      0344 2      ABBREV_FLAG = STATE_VECTOR[TPASV_ABBREV]%, ! allow minimal abbreviations
: 347      0345 2      AMBIG_FLAG = STATE_VECTOR[TPASV_AMBIG]%, ! ambiguous keyword in this state
: 348      0346 2
: 349      0347 2
: 350      0348 2 !
: 351      0349 2 ! Contents of the type byte - code and flags.
: 352      0350 2
: 353      0351 2 MACRO
: 354      0352 2      TYPECODE = 0,0,9,0%, ! full token type code
: 355      0353 2      TYPEBYTE = 0,0,8,0%, ! token type byte
: 356      0354 2      OPTION_BITS = 0,11,6,0%, ! transition option flags
: 357      0355 2      CODEFLAG = 0,8,1,0%, ! transition is special code
: 358      0356 2      EXTRAFLAG = 0,9,1,0%, ! extra type byte present
: 359      0357 2      LASTFLAG = 0,10,1,0%, ! last transition in state
: 360      0358 2      EXTFLAG = 0,11,1,0%, ! type extension present
: 361      0359 2      TRANFLAG = 0,12,1,0%, ! transition target present
: 362      0360 2      MASKFLAG = 0,13,1,0%, ! bitmask present
: 363      0361 2      ADDRFLAG = 0,14,1,0%, ! data address present
: 364      0362 2      ACTFLAG = 0,15,1,0%, ! action routine present
: 365      0363 2      PARMFLAG = 0,16,1,0%, ! parameter longword present
: 366      0364 2
: 367      0365 2 GLOBAL REGISTER
: 368      0366 2      CHAR_COUNT = 6; ! character count in string token
: 369      0367 2
: 370      0368 2 EXTERNAL LITERAL
: 371      0369 2      LIB$_SYNTAXERR, ! syntax error status
: 372      0370 2      LIB$_INVTYPE; ! invalid state table
: 373      0371 2
: 374      0372 2
: 375      0373 2 !
: 376      0374 2 !
: 377      0375 2 ! Entry initialization consists of loading the starting state

```

```
378 0376 2 | into the state pointer. Then we proceed into the main
379 0377 2 | loop that attempts to match transitions in the state table to
380 0378 2 | the current string contents.
381 0379 2 |
382 0380 2 | -
383 0381 2 |
384 0382 2 |
385 0383 2 | IF .STATE_LENGTH LSSU TPASK_COUNT0 THEN RETURN $$$_INSFARG; ! check minimum length of state vector
386 0384 2 |
387 0385 2 | AMBIG FLAG = 0;
388 0386 2 | ACT_STATUS = 0;
389 0387 2 | STATE = .START_STATE;
390 0388 2 |
391 0389 2 | WHILE 1 DO
392 0390 2 | BEGIN
393 0391 2 |
394 0392 2 | ! The following horrendous expression attempts to match the token type
395 0393 2 | of the current transition to the current string position. The outer
396 0394 2 | IF is a condition which, if true, causes the current transition
397 0395 2 | (just evaluated) to be skipped by the code at the end of this loop. The
398 0396 2 | inner IF causes the action routine of the transition to be called if
399 0397 2 | true (if the transition matches).
400 0398 2 |
401 0399 2 |
402 0400 3 | IF
403 0401 4 | BEGIN
404 0402 4 | IF
405 0403 5 | BEGIN
406 0404 5 | TYPE = GET_WORD (STATE); ! get basic type code
407 0405 5 | IF .TYPE[EXTRAFLAG] ! and extra byte if present
408 0406 5 | THEN TYPE<16,8> = GET_BYTE (STATE);
409 0407 5 |
410 0408 5 | LAST_SPACE = 0;
411 0409 5 |
412 0410 5 | IF NOT .SPACE_FLAG AND .TYPE[TYPECODE] NEQ TPAS_LAMBDA
413 0411 5 | THEN
414 0412 6 | BEGIN
415 0413 6 | GETSTRING (STRING_COUNT, SPACES);
416 0414 6 | LAST_SPACE = .CHAR_COUNT;
417 0415 6 | STRING_COUNT = .STRING_COUNT - .CHAR_COUNT; ! update string pointer
418 0416 6 | STRING_POINTER = .STRING_POINTER + .CHAR_COUNT;
419 0417 5 | END;
420 0418 5 |
421 0419 5 | CHAR_COUNT = 0; ! init matching string descriptor
422 0420 5 | TOKEN_POINTER = .STRING_POINTER;
423 0421 5 |
424 0422 5 | IF .TYPE[TYPECODE] LSSU NULL_MATCH AND .STRING_COUNT EQL 0
425 0423 5 | THEN 0 ! no match if at end
426 0424 5 | ELSE
427 0425 5 |
428 0426 5 | SELECTONEU .TYPE[TYPECODE] OF
429 0427 5 | SET
430 0428 5 |
431 0429 5 | ! Single characters are matched by token types whose numerical value is
432 0430 5 | the ASCII code of the character.
433 0431 5 |
434 0432 5 |
```

```
435 0433 5 [0 TO HIGH ASCII]: ! single ASCII character
436 0434 5 IF .TYPE[TYPEBYTE] EQL CH$RCHAR (.STRING_POINTER)
437 0435 5 THEN
438 0436 6 BEGIN
439 0437 6 STATE_CHAR = CH$RCHAR(.STRING_POINTER);
440 0438 6 CHAR_COUNT = 1
441 0439 6 END
442 0440 5 ELSE 0;
443 0441 5
444 0442 5 ! Keywords are matched by token types whose bits 0-6 contain the keyword
445 0443 5 number. A keyword token may be either (1) an exact match or
446 0444 5 (2) abbreviated to some number of characters fixed for the call or
447 0445 5 (3) arbitrarily abbreviated (such that the first match wins) or
448 0446 5 (4) abbreviated to the minimum which is locally unambiguous. Condition
449 0447 5 (4) is tested for ambiguity by finding the next entry in the keyword table
450 0448 5 and matching it against the token string. The keyword strings for each
451 0449 5 state are padded with a filler to prevent false matches across states.
452 0450 5
453 0451 5
454 0452 5
455 0453 5 [KEYWORD TO HIGH KEYWORD]: ! keyword match
456 0454 5 IF NOT .AMBIG_FLAG
457 0455 5 AND GETSTRING-(STRING_COUNT, SYMBOL)
458 0456 5 THEN
459 0457 6 BEGIN
460 0458 6 LOCAL KEY;
461 0459 6 KEY = .KEYTAB + .KEYTAB[.TYPE[TYPEBYTE]];
462 0460 6 IF CH$EQL (.CHAR_COUNT, .TOKEN_POINTER,
463 0461 6 .CHAR_COUNT, .KEY, 0)
464 0462 6 THEN
465 0463 7 BEGIN
466 0464 8 IF (CH$RCHAR (.KEY + .CHAR_COUNT) EQL TERMINATOR
467 0465 9 OR (.M$COUNT NEQ 0 AND .CHAR_COUNT GEQU .M$COUNT)
468 0466 8 )
469 0467 7 THEN 1
470 0468 7 ELSE IF .ABBRFM_FLAG
471 0469 7 THEN 1
472 0470 7 ELSE IF .ABBREV_FLAG
473 0471 7 THEN
474 0472 8 BEGIN
475 0473 8 KEY = 1 + CH$FIND CH (65535, .KEY+.CHAR_COUNT, TERMINATOR);
476 0474 8 IF CH$NEQ (.CHAR_COUNT, .TOKEN_POINTER,
477 0475 8 .CHAR_COUNT, .KEY, 0)
478 0476 8 THEN 1
479 0477 8 ELSE
480 0478 9 BEGIN
481 0479 9 AMBIG_FLAG = 1;
482 0480 9 0
483 0481 9 END
484 0482 8 END
485 0483 7 ELSE 0
486 0484 7 END
487 0485 6 ELSE 0
488 0486 6 END
489 0487 5 ELSE 0;
490 0488 5
491 0489 5 ! All other token types are special cases, representing commonly occurring
```

```
492 0490 5 : composites and other useful artifacts.
493 0491 5
494 0492 5
495 0493 5 [OTHERWISE]: ! all other types
496 0494 5 [CASE .TYPE[TYPEBYTE] FROM LOW_SPECIAL TO HIGH_SPECIAL OF
497 0495 5 SET
498 0496 5
499 0497 5 [$LAMBDA]: ! empty string
500 0498 5 1;
501 0499 5
502 0500 5 [$EOS]: ! end of input
503 0501 5 STRING_COUNT EQL 0;
504 0502 5
505 0503 5 [$ANY]: ! any single character
506 0504 6 BEGIN
507 0505 6 STATE_CHAR = CH$RCHAR (.STRING_POINTER);
508 0506 6 CHAR_COUNT = 1
509 0507 5 END;
510 0508 5
511 0509 5 [$ALPHA]: ! alphabetic
512 0510 6 IF IS_ALPHABETIC (CH$RCHAR (.STRING_POINTER))
513 0511 6 THEN (STATE_CHAR = CH$RCHAR (.STRING_POINTER);
514 0512 6 CHAR_COUNT = 1)
515 0513 5 ELSE 0;
516 0514 5
517 0515 5 [$DIGIT]: ! single digit
518 0516 6 IF IS_NUMERIC (CH$RCHAR (.STRING_POINTER))
519 0517 6 THEN (STATE_CHAR = CH$RCHAR (.STRING_POINTER);
520 0518 6 CHAR_COUNT = 1)
521 0519 5 ELSE 0;
522 0520 5
523 0521 5 [$STRING]: ! alphanumeric string
524 0522 5 GETSTRING (STRING_COUNT, ALPHANUMERIC);
525 0523 5
526 0524 5 [$SYMBOL]: ! symbol constituent set string
527 0525 5 GETSTRING (STRING_COUNT, SYMBOL);
528 0526 5
529 0527 5 [$BLANK]: ! blanks or tabs
530 0528 5 GETSTRING (STRING_COUNT, SPACES);
531 0529 5
532 0530 5 [$DECIMAL]: ! decimal number
533 0531 5 IF GETSTRING (STRING_COUNT, NUMERIC)
534 0532 5 THEN LIB$CVT_DTB (.CHAR_COUNT, .TOKEN_POINTER, STATE_NUMBER)
535 0533 5 ELSE 0;
536 0534 5
537 0535 5 [$OCTAL]: ! octal number
538 0536 5 IF GETSTRING (STRING_COUNT, NUMERIC)
539 0537 5 THEN LIB$CVT_OTB (.CHAR_COUNT, .TOKEN_POINTER, STATE_NUMBER)
540 0538 5 ELSE 0;
541 0539 5
542 0540 5 [$HEX]: ! hexa-decimal number
543 0541 5 IF GETSTRING (STRING_COUNT, HEXNUMERIC)
544 0542 5 THEN LIB$CVT_HTB (.CHAR_COUNT, .TOKEN_POINTER, STATE_NUMBER)
545 0543 5 ELSE 0;
546 0544 5
547 0545 5 [$FILESPEC, $UIC, $IDENT, $SUBEXPR]: ! complex expressions
548 0546 6 BEGIN
```

```
549 0547 6 LOCAL STATUS, SAVECOUNT, SAVEPOINTER;
550 0548 6 SAVECOUNT = .STRING_COUNT; ! save current position
551 0549 6 SAVEPOINTER = .STRING_POINTER;
552 0550 7 STATUS = (SELECTONEU .TYPE[TYPEBYTE] OF
553 0551 7 SET
554 0552 7 [$FILESPEC]:
555 0553 7 PARSE_FILESPEC (.STATE_VECTOR);
556 0554 7 [$UIC]:
557 0555 7 PARSE_IDENT (.STATE_VECTOR, 0);
558 0556 7 [$IDENT]:
559 0557 7 PARSE_IDENT (.STATE_VECTOR, 1);
560 0558 7 [$SUBEXPR]:
561 0559 7 LIB$PARSE (.STATE_VECTOR,
562 0560 7 .STATE + 2 + .(.STATE)<0,16,1>,
563 0561 7 .KEYTAB);
564 0562 6 TES);
565 0563 6 IF .STATUS NEQ LIB$ SYNTAXERR
566 0564 6 THEN ACT STATUS = .STATUS;
567 0565 6 TOKEN_POINTER = .SAVEPOINTER;
568 0566 6 CHAR_COUNT = .STRING_POINTER - .SAVEPOINTER;
569 0567 6 IF .STATUS
570 0568 6 THEN
571 0569 7 BEGIN
572 0570 7 STRING_COUNT = .STRING_COUNT + .CHAR_COUNT;
573 0571 7 STRING_POINTER = .SAVEPOINTER;
574 0572 7 1
575 0573 7 END
576 0574 6 ELSE
577 0575 7 BEGIN
578 0576 7 STRING_COUNT = .SAVECOUNT;
579 0577 7 STRING_POINTER = .SAVEPOINTER;
580 0578 7 0
581 0579 7 END
582 0580 5 END;
583 0581 5 [INRANGE]: RETURN LIB$_INVTYPE; ! just for completeness
584 0582 5 [OUTRANGE]: RETURN LIB$_INVTYPE;
585 0583 5 TES; ! end of special types case
586 0584 5
587 0585 5
588 0586 5
589 0587 5
590 0588 5 END TES ! end of select on .TYPE
591 0589 5 ! end of inner condition (token match)
592 0590 5
593 0591 5
594 0592 5
595 0593 5
596 0594 5
597 0595 5
598 0596 5
599 0597 5
600 0598 5
601 0599 5
602 0600 5
603 0601 5
604 0602 4
605 0603 5
```

The type code in this transition matches the current string, which is now described by the global string descriptor. Call the user's action routine, if it exists, and if it returns true, gobble the string and take the transition. Note that we set R0 to 1 going into the call, making it easier for the routine to return success.

```
THEN BEGIN
```

```

606 0604 5 STRING_COUNT = .STRING_COUNT - .CHAR_COUNT; ! update string pointer
607 0605 5 STRING_POINTER = .STRING_POINTER + .CHAR_COUNT;
608 0606 5 TOKEN_COUNT = .CHAR_COUNT;
609 0607 5 ! skip extension if present
610 0608 5 IF TESTBITSC (TYPE[EXTFLAG]) THEN STATE = .STATE + 2;
611 0609 5
612 0610 5 IF TESTBITSC (TYPE[PARMFLAG]) ! set parameter longword if present
613 0611 5 THEN STATE_PARAM = GET_LONG (STATE);
614 0612 5
615 0613 5 IF
616 0614 6 BEGIN
617 0615 6 IF TESTBITSC (TYPE[ACTFLAG])
618 0616 6 THEN
619 0617 7 BEGIN
620 0618 7 BUILTIN RO, CALLG;
621 0619 7 LOCAL STATUS;
622 0620 7 RO = 1;
623 0621 7 STATUS = CALLG (.STATE_VECTOR, REL_LONG (STATE)); ! call action routine
624 0622 7 IF .STATUS NEQ 0
625 0623 7 THEN ACT_STATUS = .STATUS;
626 0624 7 .STATUS
627 0625 7 END
628 0626 6 ELSE 1
629 0627 6 END
630 0628 6
631 0629 6 : Either there was no action routine, or the action routine has returned
632 0630 6 : success; we take the transition. First we get the data address, if present.
633 0631 6 : If present, store whatever data is called for: the mask, if supplied, or
634 0632 6 : type dependent data if not - either the matching character, the number
635 0633 6 : value, or the string descriptor of the matching string.
636 0634 6 :
637 0635 6 :
638 0636 5 THEN
639 0637 6 BEGIN
640 0638 6 IF .TYPE[ADDRFLAG]
641 0639 6 THEN
642 0640 7 BEGIN
643 0641 7 LOCAL ADDRESS;
644 0642 7 ADDRESS = REL_LONG (STATE);
645 0643 7 IF .TYPE[MASKFLAG]
646 0644 7 THEN
647 0645 8 .ADDRESS = ..ADDRESS OR GET_LONG (STATE)
648 0646 7 ELSE
649 0647 8 BEGIN
650 0648 8 IF NOT .TYPE[CODEFLAG]
651 0649 8 THEN
652 0650 8 (.ADDRESS)<0,8> = .STATE_CHAR
653 0651 8 ELSE
654 0652 9 BEGIN
655 0653 9 CASE .TYPE[TYPEBYTE] FROM LOW_SPECIAL TO HIGH_SPECIAL OF
656 0654 9 SET
657 0655 9 [$ANY, $ALPHA, $DIGIT]:
658 0656 9 (.ADDRESS)<0,8> = .STATE_CHAR;
659 0657 9 [$DECIMAL, $OCTAL, $HEX, $UIC, $IDENT]:
660 0658 9 .ADDRESS = .STATE_NUMBER;
661 0659 9
662 0660 9

```

```

: 663      0661  9
: 664      0662  9
: 665      0663 10
: 666      0664 10
: 667      0665 10
: 668      0666  9
: 669      0667  9
: 670      0668  9
: 671      0669  8
: 672      0670  7
: 673      0671  6
: 674      0672  6
: 675      0673  6
: 676      0674  6
: 677      0675  6
: 678      0676  6
: 679      0677  6
: 680      0678  6
: 681      0679  6
: 682      0680  6
: 683      0681  6
: 684      0682  6
: 685      0683  6
: 686      0684  6
: 687      0685  6
: 688      0686  6
: 689      0687  7
: 690      0688  6
: 691      0689  6
: 692      0690  7
: 693      0691  7
: 694      0692  7
: 695      0693  7
: 696      0694  7
: 697      0695  7
: 698      0696  6
: 699      0697  6
: 700      0698  6
: 701      0699  6
: 702      0700  6
: 703      0701  6
: 704      0702  6
: 705      0703  5
: 706      0704  6
: 707      0705  6
: 708      0706  6
: 709      0707  6
: 710      0708  6
: 711      0709  6
: 712      0710  5
: 713      0711  5
: 714      0712  4
: 715      0713  4
: 716      0714  4
: 717      0715  4
: 718      0716  4
: 719      0717  4

      [INRANGE, OTRANGE]:
      BEGIN
      (.ADDRESS) = .TOKEN_COUNT;
      (.ADDRESS+4) = .TOKEN_POINTER;
      END;

      TES;
      END;
    END;

! Take the transition. If an explicit target exists, follow it. -1 means
! exit with success, -2 means exit with failure. If no explicit target exists.
! skip transitions until we reach the beginning of the next state.
    AMBIG FLAG = 0;
    ACT_STATUS = 0;
    IF .TYPE[TRANFLAG]
    THEN
      ! take the transition
      IF .(.STATE)<0,16,1> EQL TPAS_EXIT ! TPAS_EXIT means exit
      THEN RETURN 1
      ELSE IF .(.STATE)<0,16,1> EQL TPAS_FAIL ! TPAS_FAIL means exit
      THEN RETURN LIB$_SYNTAXERR
    ELSE STATE = REL_WORD (STATE)
    ! default to next state
    UNTIL .TYPE[LASTFLAG] DO
      BEGIN
      TYPE = GET_WORD (STATE);
      IF .TYPE[EXTRAFLAG]
      THEN TYPE<16,8> = GET_BYTE (STATE);
      ! skip optional components
      STATE = .STATE + 2 * .LENGTH_TABLE[.TYPE[OPTION_BITS]];
      END;
    0
    END

! The action routine has rejected the transition. Make like it never matched.
    ELSE
    BEGIN
    ! return characters to string
    STRING_COUNT = .STRING_COUNT + .CHAR_COUNT;
    STRING_POINTER = .STRING_POINTER - .CHAR_COUNT;
    ! skip the rest of the transition
    1
    END
  END

ELSE
  1
  END
! end of outer condition
!+

```

: 9
: 9
: 9

: Rc


```

: 720 0718 4 ! If the transition does not match, we execute this code. It skips
: 721 0719 4 ! the current transition to set up to try the next one in the state.
: 722 0720 4 ! If this was the last transition in the state, a syntax error has
: 723 0721 4 ! occurred and TPARSE returns the value 0.
: 724 0722 4 !-
: 725 0723 4 !-
: 726 0724 4 !-
: 727 0725 3 THEN
: 728 0726 4 BEGIN
: 729 0727 4 STATE = .STATE + 2 * .LENGTH_TABLE[.TYPE[OPTION_BITS]];
: 730 0728 4 IF .TYPE[LASTFLAG]
: 731 0729 4 THEN
: 732 0730 5 BEGIN
: 733 0731 5 GETSTRING (STRING_COUNT, SPACES);
: 734 0732 5 GETSTRING (STRING_COUNT, SYMBOL);
: 735 0733 5 TOKEN_COUNT = .CHAR_COUNT;
: 736 0734 5 IF .TOKEN_COUNT EQL 0 AND .STRING_COUNT NEQ 0
: 737 0735 5 THEN TOKEN_COUNT = .TOKEN_COUNT + 1;
: 738 0736 6 RETURN (
: 739 0737 6 IF .ACT_STATUS[ST$V_COND_ID] NEQ 0
: 740 0738 6 THEN .ACT_STATUS
: 741 0739 5 ELSE LIB$_SYNTAXERR);
: 742 0740 4 END;
: 743 0741 4
: 744 0742 4 STRING_COUNT = .STRING_COUNT + .LAST_SPACE; ! return flushed spaces
: 745 0743 4 STRING_POINTER = .STRING_POINTER - .LAST_SPACE;
: 746 0744 4 LAST_SPACE = 0;
: 747 0745 4 END
: 748 0746 4
: 749 0747 3 END ! end of outside loop
: 750 0748 3
: 751 0749 1 END; ! end of routine TPARSE

```

```

.TITLE LIB$TPARSE
.IDENT \V04-000\
.PSECT _LIB$CODE,NOWRT, SHR, PIC,2

```

10	00000	P.AAA:	.BYTE	16
21	00001		.BYTE	33
32	00002		.BYTE	50
43	00003		.BYTE	67
32	00004		.BYTE	50
43	00005		.BYTE	67
54	00006		.BYTE	84
65	00007		.BYTE	101
32	00008		.BYTE	50
43	00009		.BYTE	67
54	0000A		.BYTE	84
65	0000B		.BYTE	101
54	0000C		.BYTE	84
65	0000D		.BYTE	101
76	0000E		.BYTE	118
87	0000F		.BYTE	-121
32	00010		.BYTE	50
43	00011		.BYTE	67

```

54 00012 .BYTE 84
65 00013 .BYTE 101
54 00014 .BYTE 84
65 00015 .BYTE 101
76 00016 .BYTE 118
87 00017 .BYTE -121
54 00018 .BYTE 84
65 00019 .BYTE 101
76 0001A .BYTE 118
87 0001B .BYTE -121
76 0001C .BYTE 118
87 0001D .BYTE -121
98 0001E .BYTE -104
A9 0001F .BYTE -87
    
```

```

LENGTH_TABLE= P.AAA
.EXTRN LIB$CVT_DTB, LIB$CVT_OTB
.EXTRN LIB$CVT_HTB, LIB$_SYNTAXERR
.EXTRN LIB$_INVTYPÉ
    
```

```

OFFC 00000 .ENTRY LIB$TPARSE, Save R2,R3,R4,R5,R6,R7,R8,R9,- R10,R11 : 0262
SE 08 C2 00002 SHBL2 #8, SP
50 04 AC D0 00005 MOVL STATE_VECTOR, R0 : 0383
08 60 D1 00009 CMPL (R0), #8
50 0114 8F 3C 0000E BGEQU 1$
06 A0 04 00013 MOVZWL #276, R0
01 8A 00014 1$: RET
55 D4 00018 BICB2 #1, 6(R0) : 0385
08 AC D0 0001A 1$: CLRL ACT STATUS : 0386
58 08 AC D0 0001A MOVL START STATE, STATE : 0387
54 88 3C 0001E 2$: MOVZWL (STATE)+, TYPE : 0404
08 54 09 E1 00021 BBC #9, TYPE, 3$ : 0405
58 D6 00025 INCL STATE : 0406
54 08 FF A8 F0 00027 INSV -1(STATE), #16, #8, TYPE
04 AE D4 0002D 3$: CLRL LAST SPACE : 0408
57 04 AC D0 00030 MOVL STATE_VECTOR, R7 : 0410
6E 04 A7 9E 00034 MOVAB 4(R7), (SP)
21 00 8E E8 00038 BLBS @0(SP), 4$
09 00 ED 0003C CMPZV #0, #9, TYPE, #502
16 13 00045 BEQL 4$
7E D4 00047 CLRL -(SP) : 0413
08 A7 9F 00049 PUSHAB 8(R7)
0000V CF 02 FB 0004C CALLS #2, GETSTRING
04 AE 56 D0 00051 MOVL CHAR_COUNT, LAST_SPACE : 0414
08 A7 56 A2 00055 SUBW2 CHAR_COUNT, 8(R7) : 0415
OC A7 56 C0 00059 ADDL2 CHAR_COUNT, 12(R7) : 0416
56 D4 0005D 4$: CLRL CHAR_COUNT : 0419
5B 14 A7 9E 0005F MOVAB 20(R7), R11
59 OC A7 9E 00063 MOVAB 12(R7), R9 : 0420
68 69 D0 00067 MOVL (R9), (R11)
00001F6 8F 54 09 00 ED 0006A CMPZV #0, #9, TYPE, #502 : 0422
08 08 1E 00073 BGEQU 5$
03 12 00078 TSTW 8(R7)
02EF 31 0007A BNEQ 5$
8F 54 09 00 ED 0007D 5$: BRW 64$ : 0433
CMPZV #0, #9, TYPE, #255
    
```

: R

		00	B9	9A	00143	15\$:	MOVZBL	20(R9)	R0	0510	
	41	8F	50	91	00147		CMPB	R0	#65		
			06	1F	0014B		BLSSU	16\$			
	5A	8F	50	91	0014D		CMPB	R0	#90		
			4D	1B	00151		BLEQU	26\$			
	61	8F	50	91	00153	16\$:	CMPB	R0	#97		
			06	1F	00157		BLSSU	17\$			
	7A	8F	50	91	00159		CMPB	R0	#122		
			41	1B	0015D		BLEQU	26\$			
	CO	8F	50	91	0015F	17\$:	CMPB	R0	#192		
			06	1F	00163		BLSSU	18\$			
	CF	8F	50	91	00165		CMPB	R0	#207		
			35	1B	00169		BLEQU	26\$			
	D1	8F	50	91	0016B	18\$:	CMPB	R0	#209		
			06	1F	0016F		BLSSU	19\$			
	DD	8F	50	91	00171		CMPB	R0	#221		
			29	1B	00175		BLEQU	26\$			
	DF	8F	50	91	00177	19\$:	CMPB	R0	#223		
			06	1F	0017B		BLSSU	20\$			
	EF	8F	50	91	0017D		CMPB	R0	#239		
			1D	1B	00181		BLEQU	26\$			
	F1	8F	50	91	00183	20\$:	CMPB	R0	#241		
			0D	1F	00187		BLSSU	22\$			
	FD	8F	50	91	00189		CMPB	R0	#253		
			0F	11	0018D		BRB	25\$			
	50		00	B9	9A	0018F	21\$:	MOVZBL	20(R9)	R0	0516
	30			50	91	00193		CMPB	R0	#48	
				03	1E	00196	22\$:	BGEQU	24\$		
				01D1	31	00198	23\$:	BRW	64\$		
		39		50	91	0019B	24\$:	CMPB	R0	#57	
				F8	1A	0019E	25\$:	BGTRU	23\$		
	18	A7		50	DD	001A0	26\$:	MOVL	R0	24(R7)	0517
		56		01	DD	001A4	27\$:	MOVL	#1	CHAR_COUNT	0518
				6C	11	001A7	28\$:	BRB	37\$		
				03	DD	001A9	29\$:	PUSHL	#3		0522
				06	11	001AB		BRB	32\$		
				04	DD	001AD	30\$:	PUSHL	#4		0525
				02	11	001AF		BRB	32\$		
				7E	D4	001B1	31\$:	CLRL	-(SP)		0528
	0000V	CF	08	A7	9F	001B3	32\$:	PUSHAB	8(R7)		
				02	FB	001B6		CALLS	#2	GETSTRING	
				55	11	001BB		BRB	36\$		
				01	DD	001BD	33\$:	PUSHL	#1		0531
	0000V	CF	08	A7	9F	001BF		PUSHAB	8(R7)		
		CE		02	FB	001C2		CALLS	#2	GETSTRING	
				50	E9	001C7		BLBC	R0	23\$	
				1C	A7	9F	001CA	PUSHAB	28(R7)		0532
				6B	DD	001CD		PUSHL	(R11)		
	00000000G	00		56	DD	001CF		PUSHL	CHAR_COUNT		
				03	FB	001D1		CALLS	#3	[IB\$CVT_DTB	
				38	11	001D8		BRB	36\$		
				01	DD	001DA	34\$:	PUSHL	#1		0536
	0000V	CF	08	A7	9F	001DC		PUSHAB	8(R7)		
		B1		02	FB	001DF		CALLS	#2	GETSTRING	
				50	E9	001E4		BLBC	R0	23\$	
				1C	A7	9F	001E7	PUSHAB	28(R7)		0537
				6B	DD	001EA		PUSHL	(R11)		

18	20	A7	88	DO	002A3		MOVL	(STATE)+, 32(R7)	0611
		54	0F	E5	002A7	50\$:	BBCC	#15, TYPE, 52\$	0615
		50	01	DO	002AB		MOVL	#1, R0	0620
		58	04	CO	002AE		ADDL2	#4, STATE	0621
	FC	B848	67	FA	002B1		CALLG	(R7), 2-4(STATE)[STATE]	
			50	DS	002B6		TSTL	STATUS	0622
			03	13	002B8		BEQL	51\$	
		55	50	DO	002BA		MOVL	STATUS, ACT_STATUS	0623
		03	50	EB	002BD	51\$:	BLBS	STATUS, 52\$	0624
			0C9D	31	002CO		BRW	63\$	
	48	54	0E	E1	002C3	52\$:	BBC	#14, TYPE, 58\$	0638
	51	58	68	C1	002C7		ADDL3	(STATE)+, STATE, ADDRESS	0642
	05	54	0D	E1	002CB		BBC	#13, TYPE, 53\$	0643
		61	88	CB	002CF		BISL2	(STATE)+, (ADDRESS)	0645
			3B	11	002D2		BRB	58\$	
		50	04	AC	002D4	53\$:	MOVL	STATE VECTOR, R0	0650
	29	54	08	E1	002D8		BBC	#8, TYPE, 56\$	0648
	0E	8F	54	8F	002DC		CASEB	TYPE, #234, #14	0653
0024	002A	002A	001E		002E1	54\$:	.WORD	55\$-54\$, -	
001E	C01E	0024	0024		002E9			57\$-54\$, -	
002A	002A	002A	001E		002F1			57\$-54\$, -	
001E	001E	001E	001E		U02F9			56\$-54\$, -	
								56\$-54\$, -	
								56\$-54\$, -	
								55\$-54\$, -	
								55\$-54\$, -	
								57\$-54\$, -	
								57\$-54\$, -	
								57\$-54\$, -	
								55\$-54\$, -	
								55\$-54\$, -	
								55\$-54\$	
		61	10	A0	7D 002FF	55\$:	MOVQ	16(R0), (ADDRESS)	0664
				0A	11 00303		BRB	58\$	0653
		61	18	A0	90 00305	56\$:	MOVQ	24(R0), (ADDRESS)	0657
				04	11 00309		BRB	58\$	
		61	1C	A0	DO 0030B	57\$:	MOVL	28(R0), (ADDRESS)	0660
		50	04	AC	DO 0030F	58\$:	MOVL	STATE VECTOR, R0	0671
				01	8A 00313		BICB2	#1, 6(R0)	0678
				55	D4 00317		CLRL	ACT_STATUS	0679
	1B	54		0C	E1 00319		BBC	#12, TYPE, 61\$	0680
		8F		68	B1 0031D		CMPW	(STATE), #-1	0682
				04	12 00322		BNEQ	59\$	
		50		01	DO 00324		MOVL	#1, 40	0683
					04 00327		RET		
		8F		68	B1 00328	59\$:	CMPW	(STATE), #-2	0684
				7F	13 0032D		BEQL	66\$	
		50		88	32 0032F		C'TWL	(STATE)+, R0	0687
		58		50	CO 00332		ADDL2	R0, STATE	
				FCE6	31 00335	60\$:	BRW	2\$	0682
	F9	54		0A	E0 00338	61\$:	BBS	#10, TYPE, 60\$	0689
		54		88	3C 0033C		MOVZWL	(STATE)+, TYPE	0691
	08	54		09	E1 0033F		BBC	#9, TYPE, 62\$	0692
				58	D6 00343		INCL	STATE	0693
	54	10	FF	A8	F0 00345		INSV	-1(STATE), #16, #8, TYPE	
	51	06		0B	EF 0034B	62\$:	EXTZV	#11, #6, TYPE, R1	0695

50	FC87	CF	51	04	C4	00350	MULL2	#4, R1	0689	
			04	51	EF	00353	EXTZV	R1, #4, LENGTH TABLE, R0	0704	
			58	6840	3E	0035A	MOVAV	(STATE)[R0], STATE	0705	
				08	04	AC	DO	00360 63\$:	0706	
				0C	56	A0	00364	ADDW2	CHAR_COUNT, 8(R0)	
					56	C2	00368	SUBL2	CHAR_COUNT, 12(R0)	
51		54	06	08	EF	0036C	64\$:	EXTZV	#11, #6, TYPE, R1	
			51	04	C4	00371	MULL2	#4, R1	0727	
50	FC66	CF	04	51	EF	00374	EXTZV	R1, #4, LENGTH TABLE, R0		
			58	6840	3E	0037B	MOVAV	(STATE)[R0], STATE		
		39	54	0A	E1	0037F	BBC	#10, TYPE, 68\$	0728	
					7E	D4	00383	CLRL	-(SP)	
			52	04	AC	DO	00385	MOVL	STATE_VECTOR, R2	
				08	A2	9F	00389	PUSHAB	8(R2)	
			0000V	CF	02	FB	0038C	CALLS	#2, GETSTRING	
					04	DD	00391	PUSHL	#4	
					08	A2	9F	00393	PUSHAB	8(R2)
			0000V	CF	02	FB	00396	CALLS	#2, GETSTRING	
			10	A2	56	D0	0039B	MOVL	CHAR_COUNT, 16(R2)	
					08	12	0039F	BNEQ	65\$	
					08	A2	B5	003A1	TSTW	8(R2)
					03	13	003A4	BEQL	65\$	
				10	A2	D6	003A6	INCL	16(R2)	
00		55	19	03	ED	003A9	65\$:	CMPZV	#3, #25, ACT_STATUS, #0	
				04	13	003AE	66\$:	BEQL	67\$	
			50	55	D0	003B0	MOVL	ACT_STATUS, R0	0738	
					04	003B3	RET			
			50	00000000G	8F	D0	003B4	67\$:	MOVL	#LIB\$_SYNTAXERR, R0
					04	003BB	RET		0736	
					04	AC	DO	003BC	68\$:	
			08	A0	04	AE	A0	003C0	ADDW2	LAST_SPACE, 8(R0)
			0C	A0	04	AE	C2	003C5	SUBL2	LAST_SPACE, 12(R0)
					04	AE	D4	003CA	CLRL	LAST_SPACE
					FC4E	31	003CD	BRW	2\$	

; Routine Size: 976 bytes, Routine Base: _LIB\$CODE + 0020

10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

; R

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

: 753 0750 1 ROUTINE GETSTRING (STRING, TYPE) : L_GETSTRING =
: 754 0751 1
: 755 0752 1 !++
: 756 0753 1
: 757 0754 1 FUNCTIONAL DESCRIPTION:
: 758 0755 1
: 759 0756 1 This routine extracts a string of the indicated type from
: 760 0757 1 the front of the string being parsed.
: 761 0758 1
: 762 0759 1
: 763 0760 1 CALLING SEQUENCE:
: 764 0761 1 GETSTRING (ARG1, ARG2)
: 765 0762 1
: 766 0763 1
: 767 0764 1 INPUT PARAMETERS:
: 768 0765 1 ARG1 = address of string descriptor of source string
: 769 0766 1 ARG2 = string type code
: 770 0767 1
: 771 0768 1 IMPLICIT INPUTS:
: 772 0769 1 NONE
: 773 0770 1
: 774 0771 1 OUTPUT PARAMETERS:
: 775 0772 1 NONE
: 776 0773 1
: 777 0774 1 IMPLICIT OUTPUTS:
: 778 0775 1 NONE
: 779 0776 1
: 780 0777 1 ROUTINE VALUE:
: 781 0778 1 1 if string is not empty
: 782 0779 1 0 if string is null
: 783 0780 1
: 784 0781 1 SIDE EFFECTS:
: 785 0782 1 NONE
: 786 0783 1
: 787 0784 1 --
: 788 0785 1
: 789 0786 1
: 790 0787 2 BEGIN
: 791 0788 2
: 792 0789 2 MAP
: 793 0790 2 STRING: REF VECTOR;
: 794 0791 2
: 795 0792 2 EXTERNAL REGISTER
: 796 0793 2 CHAR_COUNT = 6; ! character count of found string
: 797 0794 2
: 798 0795 2 !+
: 799 0796 2
: 800 0797 2 To extract the string we simply scan through the input string
: 801 0798 2 until we hit a character that is not in the class.
: 802 0799 2 We count the characters in the global string count.
: 803 0800 2
: 804 0801 2 !-
: 805 0802 2
: 806 0803 2 CHAR_COUNT = 0;
: 807 0804 2
: 808 0805 2 WHILE .(STRING[0])<0,16> GTRU .CHAR_COUNT
: 809 0806 2 AND (
```

: 1
: 1
: 1
: 1
: 1

: R


```

810 0807 4 ( IF .TYPE EQL SPACES
811 0808 5 THEN IS_SPACE (CH$RCHAR (.STRING[1] + .CHAR_COUNT))
812 0809 4 ELSE 0)
813 0810 4 OR (IF .TYPE GEQU NUMERIC
814 0811 5 THEN IS_NUMERIC (CH$RCHAR (.STRING[1] + .CHAR_COUNT))
815 0812 4 ELSE 0)
816 0813 4 OR (IF .TYPE GEQU HEXNUMERIC
817 0814 5 THEN IS_HEX (CH$RCHAR (.STRING[1] + .CHAR_COUNT))
818 0815 4 ELSE 0)
819 0816 4 OR (IF .TYPE GEQU ALPHANUMERIC
820 0817 5 THEN IS_ALPHABETIC (CH$RCHAR (.STRING[1] + .CHAR_COUNT))
821 0818 4 ELSE 0)
822 0819 4 OR (IF .TYPE EQL SYMBOL
823 0820 5 THEN IS_SYMBOL (CH$RCHAR (.STRING[1] + .CHAR_COUNT))
824 0821 4 ELSE 0)
825 0822 3 )
826 0823 2 DO (CHAR_COUNT = .CHAR_COUNT + 1);
827 0824 2
828 0825 2 RETURN .CHAR_COUNT GTRU 0;
829 0826 2
830 0827 1 END;

```

! end of routine GETSTRING

				0004 00000 GETSTRING:				
					.WORD	Save R2		0750
					CLRL	CHAR_COUNT		0803
56		51	04	AC D0 00004	MOVL	STRING, R1		0805
	61	10		00 ED 00008 1\$:	CMPZV	#0, #16, (R1), CHAR_COUNT		
				03 1A 0000D	BGTRU	2\$		
				00C0 31 0000F	BRW	17\$		
		52	08	AC D0 00012 2\$:	MOVL	TYPE, R2		0807
				0F 12 00016	BNEQ	3\$		
		50	04	B146 9A 00018	MOVZBL	@4(R1)[CHAR_COUNT], R0		0808
		09		50 91 0001D	CMPB	R0, #9		
				7C 13 00020	BEQL	11\$		
		20		50 91 00022	CMPB	R0, #32		
				77 13 00025	BEQL	11\$		
				52 D5 00027 3\$:	TSTL	R2		0810
				0F 13 00029	BEQL	4\$		
		50	04	B146 9A 0002B	MOVZBL	@4(R1)[CHAR_COUNT], R0		0811
		30		50 91 00030	CMPB	R0, #48		
				05 1F 00033	BLSSU	4\$		
		39		50 91 00035	CMPB	R0, #57		
				7C 1B 00038	BLEQU	14\$		
		02		52 D1 0003A 4\$:	CMPL	R2, #2		0813
				27 1F 0003D	BLSSU	7\$		
		50	04	B146 9A 0003F	MOVZBL	@4(R1)[CHAR_COUNT], R0		0814
		30		50 91 00044	CMPB	R0, #48		
				05 1F 00047	BLSSU	5\$		
		39		50 91 00049	CMPB	R0, #57		
				7F 1B 0004C	BLEQU	16\$		
		41	8F	50 91 0004E 5\$:	CMPB	R0, #65		
				06 1F 00052	BLSSU	6\$		
		46	8F	50 91 00054	CMPB	R0, #70		

61	8F	73	1B	00058	BLEQU	16\$			
		50	91	0005A	6\$: CMPB	RO, #97			
		06	1F	0005E	BLSSU	7\$			
66	8F	50	91	00060	CMPB	RO, #102			
		67	1B	00064	BLEQU	16\$			
	03	52	D1	00066	7\$: CMPL	R2, #3		0816	
		4D	1F	00069	BLSSU	15\$			
	50	04 B146	9A	0006B	MOVZBL	@4(R1)[CHAR_COUNT], RO		0817	
41	8F	50	91	00070	CMPB	RO, #65			
		06	1F	00074	BLSSU	8\$			
5A	8F	50	91	00076	CMPB	RO, #90			
		51	1B	0007A	BLEQU	16\$			
61	8F	50	91	0007C	8\$: CMPB	RO, #97			
		06	1F	00080	BLSSU	9\$			
7A	8F	50	91	00082	CMPB	RO, #122			
		45	1B	00086	BLEQU	16\$			
CO	8F	50	91	00088	9\$: CMPB	RO, #192			
		06	1F	0008C	BLSSU	10\$			
CF	8F	50	91	0008E	CMPB	RO, #207			
		39	1B	00092	BLEQU	16\$			
D1	8F	50	91	00094	10\$: CMPB	RO, #209			
		06	1F	00098	BLSSU	12\$			
DD	8F	50	91	0009A	CMPB	RO, #221			
		2D	1B	0009E	BLEQU	16\$			
DF	8F	50	91	000A0	12\$: CMPB	RO, #223			
		06	1F	000A4	BLSSU	13\$			
EF	8F	50	91	000A6	CMPB	RO, #239			
		21	1B	000AA	BLEQU	16\$			
F1	8F	50	91	000AC	13\$: CMPB	RO, #241			
		06	1F	000B0	BLSSU	15\$			
FD	8F	50	91	000B2	CMPB	RO, #253			
		15	1B	000B6	14\$: BLEQU	16\$			
	04	52	D1	000B8	15\$: CMPL	R2, #4		0819	
		15	12	000BB	BNEQ	17\$			
	50	04 B146	9A	000BD	MOVZBL	@4(R1)[CHAR_COUNT], RO		0820	
	24	50	91	000C2	CMPB	RO, #36			
		06	13	000C5	BEQL	16\$			
SF	8F	50	91	000C7	CMPB	RO, #95			
		05	12	000CB	BNEQ	17\$			
		56	D6	000CD	16\$: INCL	CHAR_COUNT		0823	
		FF36	31	000CF	BRW	1\$			
		50	D4	000D2	17\$: CLRL	RO		0825	
		56	D5	000D4	TSTL	CHAR_COUNT			
		02	13	000D6	BEQL	18\$			
		50	D6	000D8	INCL	RO			
		04	000DA	18\$: RET				0827	

; Routine Size: 219 bytes, Routine Base: _LIB\$CODE + 03F0

```
.. 832      0828 1  |
.. 833      0829 1  | Complex type parsing routines. The following routines parse complex
.. 834      0830 1  | token types by recursing through TPARSE, using local parse tables for
.. 835      0831 1  | the individual types.
.. 836      0832 1  |
.. 837      0833 1  |
.. 838      0834 1  |
.. 839      0835 1  | Declare TPARSE block passed as arg list.
.. 840      0836 1  |
.. 841      0837 1  |
.. 842      0838 1  | MACRO
.. 843      0839 1  |     TPARSE_ARGS =
.. 844      0840 1  |         BUILTIN AP:
.. 845      0841 1  |         BIND TPARSE_BLOCK = AP : REF $BLOCK;
.. 846      0842 1  |         %;
.. 847      0843 1  |
.. 848      0844 1  |
.. 849      0845 1  | | TPARSE block extension usage by following routines:
.. 850      0846 1  |
.. 851      0847 1  |
.. 852      0848 1  | MACRO
.. 853      0849 1  |     TEMP_NUMBER      = TPASK_LENGTH0+00, 0, 32, 0%;
.. 854      0850 1  |     TEMP_CHAR        = TPASK_LENGTH0+04, 0, 8, 0%;
.. 855      0851 1  |
.. 856      0852 1  | LITERAL
.. 857      0853 1  |     LOCAL_STORAGE    = 8;
.. 858      0854 1  |
.. 859      0855 1  | FORWARD
.. 860      0856 1  |     LIB$$UIC_STATE_TABLE : VECTOR [0],
.. 861      0857 1  |     LIB$$UIC_KEY_TABLE   : VECTOR [0],
.. 862      0858 1  |     LIB$$IDENT_STATE_TABLE : VECTOR [0],
.. 863      0859 1  |     LIB$$IDENT_KEY_TABLE : VECTOR [0];
```

```
865 0860 1 ROUTINE PARSE_FILESPEC (TPARSE_BLOCK) =
866 0861 1
867 0862 1 ++
868 0863 1
869 0864 1 FUNCTIONAL DESCRIPTION:
870 0865 1
871 0866 1 This routine parses a file specifier string, using the $FILESCAN
872 0867 1 service.
873 0868 1
874 0869 1 CALLING SEQUENCE:
875 0870 1 PARSE_FILESPEC (TPARSE_BLOCK)
876 0871 1
877 0872 1 INPUT PARAMETERS:
878 0873 1 TPARSE_BLOCK: address of TPARSE control block
879 0874 1
880 0875 1 IMPLICIT INPUTS:
881 0876 1 NONE
882 0877 1
883 0878 1 OUTPUT PARAMETERS:
884 0879 1 NONE
885 0880 1
886 0881 1 IMPLICIT OUTPUTS:
887 0882 1 NONE
888 0883 1
889 0884 1 ROUTINE VALUE:
890 0885 1 status of parse operation
891 0886 1
892 0887 1 SIDE EFFECTS:
893 0888 1 string descriptors in TPARSE block modified if successful
894 0889 1
895 0890 1 --
896 0891 1
897 0892 2 BEGIN
898 0893 2
899 0894 2 MAP
900 0895 2 TPARSE_BLOCK : REF $BBLOCK; ! TPARSE control block arg
901 0896 2
902 0897 2 LOCAL
903 0898 2 FSCAN_LIST : $BBLOCK [12]; ! descriptor list for $FILESCAN
904 0899 2
905 0900 2
906 0901 2 ! Set up the filescan list arguments and call it.
907 0902 2 !
908 0903 2
909 0904 2 FSCAN_LIST[ATR$W_TYPE] = FSCAN$_FILESPEC;
910 0905 2 FSCAN_LIST+8 = 0;
911 P 0906 2 IF NOT $FILESCAN (SRCSTR = TPARSE_BLOCK[TPAS$L_STRINGCNT],
912 0907 2 VALU$ST = FSCAN_LIST)
913 0908 2 THEN RETURN 0;
914 0909 2
915 0910 2 ! If successful, update the string pointers in the user's block.
916 0911 2 !
917 0912 2
918 0913 2 TPARSE_BLOCK[TPAS$L_STRINGCNT] = .TPARSE_BLOCK[TPAS$L_STRINGCNT]
919 0914 2 - .FSCAN_LIST[ATR$W_SIZE];
920 0915 2 TPARSE_BLOCK[TPAS$L_STRINGPTR] = .TPARSE_BLOCK[TPAS$L_STRINGPTR]
921 0916 2 + .FSCAN_LIST[ATR$W_SIZE];
```

: 922 0917 2
: 923 0918 2 1
: 924 0919 1 END;

! End of routine PARSE_FILESPEC

		.EXTRN SYSS\$FILESCAN		
		0004 00000	PARSE_FILESPEC:	
			.WORD	Save R2
			SUBL2	#12, SP
02	5E	0C C2 00002	MOVW	#1, FSCAN_LIST+2
	AE	01 B0 00005	CLRL	FSCAN_LIST+8
		08 AE D4 00009	CLRL	-(SP)
		7E D4 0000C	PUSHAB	FSCAN_LIST
		04 AE 9F 0000E	MOVL	TPARSE_BLOCK, R2
	52	04 AC D0 00011	PUSHAB	8(R2)
		08 A2 9F 00015	CALLS	#3, SYSS\$FILESCAN
00000000G	00	03 FB 00018	BLBS	R0, 1\$
	03	50 E8 0001F	CLRL	R0
		50 D4 00022	RET	
		04 00024	MOVZWL	FSCAN_LIST, R0
	50	6E 3C 00025 1\$:	SUBL2	R0, 8(R2)
08	A2	50 C2 00028	MOVZWL	FSCAN_LIST, R0
	50	6E 3C 0002C	ADDL2	R0, 12(R2)
0C	A2	50 C0 0002F	MOVL	#1, R0
	50	01 D0 00033	RET	
		04 00036		

; Routine Size: 55 bytes, Routine Base: _LIB\$CODE + 04CB

```

: 926 0920 1 ROUTINE PARSE_IDENT (TPARSE_BLOCK, GENERAL) =
: 927 0921 1
: 928 0922 1 ++
: 929 0923 1
: 930 0924 1 FUNCTIONAL DESCRIPTION:
: 931 0925 1
: 932 0926 1 This routine parses an identifier string, leaving the binary value
: 933 0927 1 in the TPASL_NUMBER field of the tparse block.
: 934 0928 1
: 935 0929 1 CALLING SEQUENCE:
: 936 0930 1 PARSE_IDENT (TPARSE_BLOCK, GENERAL)
: 937 0931 1
: 938 0932 1 INPUT PARAMETERS:
: 939 0933 1 TPARSE_BLOCK: address of TPARSE control block
: 940 0934 1 GENERAL: 0 to parse UIC only
: 941 0935 1 1 to parse general identifier
: 942 0936 1
: 943 0937 1 IMPLICIT INPUTS:
: 944 0938 1 NONE
: 945 0939 1
: 946 0940 1 OUTPUT PARAMETERS:
: 947 0941 1 NONE
: 948 0942 1
: 949 0943 1 IMPLICIT OUTPUTS:
: 950 0944 1 NONE
: 951 0945 1
: 952 0946 1 ROUTINE VALUE:
: 953 0947 1 status of parse operation
: 954 0948 1
: 955 0949 1 SIDE EFFECTS:
: 956 0950 1 string descriptors in TPARSE block modified if successful
: 957 0951 1
: 958 0952 1 --
: 959 0953 1
: 960 0954 2 BEGIN
: 961 0955 2
: 962 0956 2 MAP
: 963 0957 2 TPARSE_BLOCK : REF $BBLOCK; ! TPARSE control block arg
: 964 0958 2
: 965 0959 2 LOCAL
: 966 0960 2 STATUS, ! general status value
: 967 0961 2 LOCAL_BLOCK : $BBLOCK [TPASK_LENGTH0+LOCAL_STORAGE];
: 968 0962 2 ! local TPARSE control block
: 969 0963 2
: 970 0964 2 ! Set up the local TPARSE control block by copying the user's.
: 971 0965 2 ! Then do the parse.
: 972 0966 2
: 973 0967 2
: 974 0968 2 CH$MOVE (TPASK_LENGTH0, .TPARSE_BLOCK, LOCAL_BLOCK);
: 975 0969 2 LOCAL_BLOCK[TPASL_COUNT] = TPASK_COUNT0;
: 976 0970 2 LOCAL_BLOCK[TPASV_BLANKS] = 1;
: 977 0971 2 STATUS = (
: 978 0972 3 IF .GENERAL
: 979 0973 3 THEN LIB$TPARSE (LOCAL_BLOCK, LIB$$IDENT_STATE_TABLE, LIB$$IDENT_KEY_TABLE)
: 980 0974 3 ELSE LIB$TPARSE (LOCAL_BLOCK, LIB$$UIC_STATE_TABLE, LIB$$UIC_KEY_TABLE)
: 981 0975 2 );
: 982 0976 2
```

```

: 983      0977 2 ! If successful, update the string pointers in the user's block.
: 984      0978 2 !
: 985      0979 2 !
: 986      0980 2 IF .STATUS
: 987      0981 2 THEN
: 988      0982 2 BEGIN
: 989      0983 2 TPARSE_BLOCK[TPAS$L_STRINGCNT] = .LOCAL_BLOCK[TPAS$L_STRINGCNT];
: 990      0984 2 TPARSE_BLOCK[TPAS$L_STRINGPTR] = .LOCAL_BLOCK[TPAS$L_STRINGPTR];
: 991      0985 2 TPARSE_BLOCK[TPAS$L_NUMBER] = .LOCAL_BLOCK[TPAS$L_NUMBER];
: 992      0986 2 END;
: 993      0987 2
: 994      0988 2 .STATUS
: 995      0989 1 END;
! End of routine PARSE_IDENT

```

				007C 00000 PARSE_IDENT:				
		5E	2C	C2	00002	.WORD	Save R2,R3,R4,R5,R6	: 0920
		56	AC	D0	00005	SUBL2	#44, SP	: 0968
6E		66	24	28	00009	MOVL	TPARSE_BLOCK, R6	: 0969
		6E	08	D0	0000D	MOVC3	#36, (R6), LOCAL_BLOCK	: 0970
	04	AE	01	88	00010	MOVL	#8, LOCAL_BLOCK	: 0972
		0A	08	AC	E9 00014	BISB2	#1, LOCAL_BLOCK+4	: 0973
			0000V	CF	9F 00018	BLBC	GENERAL, T\$: 0974
			0000V	CF	9F 0001C	PUSHAB	LIB\$\$IDENT_KEY_TABLE	: 0975
				08	11 00020	PUSHAB	LIB\$\$IDENT_STATE_TABLE	: 0976
			0000V	CF	9F 00022 1\$:	BRB	2\$: 0977
			0000V	CF	9F 00026	PUSHAB	LIB\$\$SUIC_KEY_TABLE	: 0978
			08	AE	9F 0002A 2\$:	PUSHAB	LIB\$\$SUIC_STATE_TABLE	: 0979
	FAEC	CF		03	FB 0002D	PUSHAB	LOCAL_BLOCK	: 0980
		0A		50	E9 00032	CALLS	#3, LIB\$TPARSE	: 0981
	08	A6	08	AE	7D 00035	BLBC	STATUS, 3\$: 0982
	1C	A6	1C	AE	D0 0003A	MOVQ	LOCAL_BLOCK+8, 8(R6)	: 0983
				04	0003F 3\$:	MOVL	LOCAL_BLOCK+28, 28(R6)	: 0984
						RET		: 0989

; Routine Size: 64 bytes, Routine Base: _LIB\$CODE + 0502

```

: 997      0990 1 |
: 998      0991 1 | Action routines to parse UIC's and identifiers.
: 999      0992 1 |
: 1000     0993 1 |
: 1001     0994 1 |
: 1002     0995 1 | Remember opening bracket.
: 1003     0996 1 |
: 1004     0997 1 | ROUTINE SET_BRACKET =
: 1005     0998 2 | BEGIN
: 1006     0999 2 | TPARSE_ARGS;
: 1007     1000 2 |
: 1008     1001 2 | TPARSE_BLOCK[TEMP_CHAR] = .TPARSE_BLOCK[TPASB_CHAR];
: 1009     1002 1 |
: 1010     1003 1 | END;

```

```

                                0000 00000 SET_BRACKET:
                                .WORD   Save nothing
                                28 AC    18 AC 90 00002   MOVB   24(TPARSE_BLOCK), 40(TPARSE_BLOCK)
                                01 D0 00007   MOVL  #1, R0
                                04 0000A   RET

```

```

: 0997
: 1001
: 1003
:

```

: Routine Size: 11 bytes, Routine Base: _LIB\$CODE + 0542

```

: 1011     1004 1 |
: 1012     1005 1 |
: 1013     1006 1 | Check closing bracket against opening bracket.
: 1014     1007 1 |
: 1015     1008 1 | ROUTINE CHECK_BRACKET =
: 1016     1009 2 | BEGIN
: 1017     1010 2 | TPARSE_ARGS;
: 1018     1011 2 |
: 1019     1012 2 | $ASSUME ('<'-'>', EQL, '['-'']');
: 1020     1013 3 | .TPARSE_BLOCK[TPASB_CHAR] EQL .TPARSE_BLOCK[TEMP_CHAR] + ('')-'[')
: 1021     1014 3 |
: 1022     1015 1 | END;

```

```

                                0000 00000 CHECK_BRACKET:
                                .WORD   Save nothing
                                51      28 AC 9A 00002   MOVZBL 40(TPARSE_BLOCK), R1
                                51      02 C0 00006   ADDL2  #2, R1
                                50      50 D4 00009   CLRL   R0
                                51      00 ED 0000B   CMPZV  #0, #8, 24(TPARSE_BLOCK), R1
                                02      12 00011   BNEQ   1$
                                50      D6 00013   INCL   R0
                                04 00015 1$:   RET

```

```

: 1008
: 1013
:
:
:
:
: 1015

```

: Routine Size: 22 bytes, Routine Base: _LIB\$CODE + 0540


```

: 1023      1016 1
: 1024      1017 1
: 1025      1018 1  Check and save group number.
: 1026      1019 1
: 1027      1020 1 ROUTINE STORE_GROUP =
: 1028      1021 2 BEGIN
: 1029      1022 2 TPARSE_ARGS;
: 1030      1023 2
: 1031      1024 2 (TPARSE_BLOCK[TEMP_NUMBER])<16,16> = TPARSE_BLOCK[TPASL_NUMBER];
: 1032      1025 2 .TPARSE_BLOCK[TPASL_NUMBER] LSSU 16383
: 1033      1026 2
: 1034      1027 1 END;

```

```

                                0000 00000 STORE_GROUP:
                                .WORD      Save nothing
                                MOVW      28(TPARSE_BLOCK), 38(TPARSE_BLOCK)
                                CLRL      R0
                                26 AC      1C AC B0 00002
                                50 D4 00007
                                00003FFF 8F 1C AC D1 00009
                                02 1E 00011
                                50 D6 00013
                                04 00015 1$: INCL R0
                                RET
: 1020
: 1024
: 1025
:
: 1027

```

: Routine Size: 22 bytes, Routine Base: _LIB\$CODE + 0563

```

: 1035      1028 1
: 1036      1029 1
: 1037      1030 1  Store wild card group number.
: 1038      1031 1
: 1039      1032 1 ROUTINE WILD_GROUP =
: 1040      1033 2 BEGIN
: 1041      1034 2 TPARSE_ARGS;
: 1042      1035 2
: 1043      1036 2 (TPARSE_BLOCK[TEMP_NUMBER])<16,16> = 16383;
: 1044      1037 2
: 1045      1038 2 1
: 1046      1039 1 END;

```

```

                                0000 00000 WILD_GROUP:
                                .WORD      Save nothing
                                MOVW      #16383, 38(TPARSE_BLOCK)
                                26 AC      3FFF 8F B0 00002
                                50 D0 00008
                                04 00008
                                MOVL      #1, R0
                                RET
: 1032
: 1036
: 1039
:

```

: Routine Size: 12 bytes, Routine Base: _LIB\$CODE + 0579

: 1047 1040 1

```

: 1048      1041 1  |
: 1049      1042 1  | Check and save member number.
: 1050      1043 1  |
: 1051      1044 1  | ROUTINE STORE_MEMBER =
: 1052      1045 2  | BEGIN
: 1053      1046 2  | TPARSE_ARGS;
: 1054      1047 2  |
: 1055      1048 2  | IF .TPARSE_BLOCK[TPASL_NUMBER] GEQU 65535
: 1056      1049 2  | THEN RETURN 0;
: 1057      1050 2  | (TPARSE_BLOCK[TPASL_NUMBER])<16,16> = .(TPARSE_BLOCK[TEMP_NUMBER])<16,16>;
: 1058      1051 2  |
: 1059      1052 1  |
: 1060      1053 1  | END;

```

```

                                0000 00000 STORE_MEMBER:
                                .WORD   Save nothing
0000FFFF 8F 1C AC D1 00002      CMPL   28(TPARSE_BLOCK), #65535
                                BLSSU   1$
                                03 1F 0000A      CLRL   R0
                                50 D4 0000C      RET
                                04 0000E
                                1E AC 26 AC B0 0000F 1$: MOVW   38(TPARSE_BLOCK), 30(TPARSE_BLOCK)
                                01 D0 00014      MOVL   #1, R0
                                04 00017      RET
: 1044
: 1048
: 1049
: 1050
: 1053

```

; Routine Size: 24 bytes, Routine Base: _LIB\$CODE + 0585

```

: 1061      1054 1  |
: 1062      1055 1  |
: 1063      1056 1  | Store wild card member number.
: 1064      1057 1  |
: 1065      1058 1  | ROUTINE WILD_MEMBER =
: 1066      1059 2  | BEGIN
: 1067      1060 2  | TPARSE_ARGS;
: 1068      1061 2  |
: 1069      1062 2  | (TPARSE_BLOCK[TPASL_NUMBER])<16,16> = .(TPARSE_BLOCK[TEMP_NUMBER])<16,16>;
: 1070      1063 2  | (TPARSE_BLOCK[TPASL_NUMBER])<0,16> = 65535;
: 1071      1064 2  |
: 1072      1065 2  |
: 1073      1066 1  | END;

```

```

                                0000 00000 WILD_MEMBER:
                                .WORD   Save nothing
1E AC 26 AC B0 00002      MOVW   38(TPARSE_BLOCK), 30(TPARSE_BLOCK)
1C AC 01 AE 00007      MNEGW #1, 28(TPARSE_BLOCK)
                                01 D0 0000B      MOVL   #1, R0
                                04 0000E      RET
: 1058
: 1062
: 1063
: 1066

```

; Routine Size: 15 bytes, Routine Base: _LIB\$CODE + 059D

```

: 1074      1067 1
: 1075      1068 1
: 1076      1069 1  Store full wild card identifier.
: 1077      1070 1
: 1078      1071 1 ROUTINE ALL_WILD =
: 1079      1072 2 BEGIN
: 1080      1073 2 TPARSE_ARGS;
: 1081      1074 2
: 1082      1075 2 TPARSE_BLOCK[TPASL_NUMBER] = UIC$K_MATCH_ALL;
: 1083      1076 2
: 1084      1077 1
: 1085      1078 1 END;

```

```

                                0000 00000 ALL_WILD:
                                .WORD  Save nothing
                                MNEGL #1, 28(TPARSE_BLOCK)
                                MOVL  #1, R0
                                RET
                                1C  AC      01  CE 00002
                                01  D0 00006
                                04  00009

```

```

: 1071
: 1075
: 1078
:

```

: Routine Size: 10 bytes, Routine Base: _LIB\$CODE + 05AC

```

: 1086      1079 1
: 1087      1080 1
: 1088      1081 1  Convert single identifier name to numeric value.
: 1089      1082 1
: 1090      1083 1 ROUTINE SINGLE_NAME =
: 1091      1084 2 BEGIN
: 1092      1085 2 TPARSE_ARGS;
: 1093      1086 2
: 1094      P 1087 2 $ASCTOID (NAME = TPARSE_BLOCK[TPASL_TOKENCNT],
: 1095      1088 3 ID = TPARSE_BLOCK[TPASL_NUMBER])
: 1096      1089 3
: 1097      1090 1 END;

```

```

                                .EXTRN  SYSSASCTOID
                                0000 00000 SINGLE_NAME:
                                .WORD  Save nothing
                                CLRL  -(SP)
                                PUSHAB 28(TPARSE_BLOCK)
                                PUSHAB 16(TPARSE_BLOCK)
                                CALLS  #3, SYSSASCTOID
                                RET
                                00000000G 00
                                7E  D4 00002
                                1C  AC  9F 00004
                                10  AC  9F 00007
                                03  FB 0000A
                                04  00011

```

```

: 1083
: 1088
:
: 1090

```

: Routine Size: 18 bytes, Routine Base: _LIB\$CODE + 05B6

: 1098 1091 1

```

: 1099      1092 1 |
: 1100      1093 1 | Convert and save group name.
: 1101      1094 1 |
: 1102      1095 1 ROUTINE GROUP_NAME =
: 1103      1096 2 BEGIN
: 1104      1097 2 TPARSE_ARGS;
: 1105      1098 2
: 1106      1099 2 LOCAL
: 1107      1100 2 STATUS;
: 1108      1101 2
: 1109      P 1102 2 STATUS = $ASCTOID (NAME = TPARSE_BLOCK[TPASL_TOKENCNT],
: 1110      1103 2 ID = TPARSE_BLOCK[TEMP_NUMBER]);
: 1111      1104 2 IF NOT .STATUS THEN RETURN .STATUS;
: 1112      1105 2 .(TPARSE_BLOCK[TEMP_NUMBER])<16,16> LEQU 16383 AND
: 1113      1106 2 .(TPARSE_BLOCK[TEMP_NUMBER])<00,16> EQLU 65535
: 1114      1107 2
: 1115      1108 1 END;

```

		0004 0000 GROUP_NAME:				
						Save R2
		24	7E D4 00002			-(SP)
		10	AC 9F 00004			PUSHAB 36(TPARSE_BLOCK)
			AC 9F 00007			PUSHAB 16(TPARSE_BLOCK)
00000000G	00		03 FB 0000A			CALLS #3, SYSSASCTOID
	1E		50 E9 00011			BLBC STATUS, 3\$
			51 D4 00014			CLRL R1
3FFF	8F	26	AC B1 00016			CMPW 38(TPARSE_BLOCK), #16383
			02 1A 0001C			BGTRU 1\$
			51 D6 0001E			INCL R1
			50 D4 00020	1\$:		CLRL R0
FFFF	8F	24	AC B1 00022			CMPW 36(TPARSE_BLOCK), #65535
			02 12 00028			BNEQ 2\$
			50 D6 0002A			INCL R0
	52		51 D2 0002C	2\$:		MCOML R1, R2
	50		52 CA 0002F			BICL2 R2, R0
			04 00032	3\$:		RET

```

: 1095
: 1103
:
: 1104
: 1105
:
: 1106
:
: 1108

```

: Routine Size: 51 bytes, Routine Base: _LIB\$CODE + 05C8

```

: 1116      1109 1 |
: 1117      1110 1 |
: 1118      1111 1 | Convert and check member name.
: 1119      1112 1 |
: 1120      1113 1 ROUTINE MEMBER_NAME =
: 1121      1114 2 BEGIN
: 1122      1115 2 TPARSE_ARGS;
: 1123      1116 2
: 1124      1117 2 LOCAL
: 1125      1118 2 STATUS;
: 1126      1119 2
: 1127      P 1120 2 STATUS = $ASCTOID (NAME = TPARSE_BLOCK[TPASL_TOKENCNT],
: 1128      1121 2 ID = TPARSE_BLOCK[TPASL_NUMBER]);

```

```

: 1129      1122  2 IF NOT .STATUS THEN RETURN .STATUS;
: 1130      1123  2 .(TPARSE_BLOCK[TEMP_NUMBER])<16,16> EQL
: 1131      1124  2 .(TPARSE_BLOCK[TPARSE_NUMBER])<16,16>
: 1132      1125  2
: 1133      1126  1 END;

```

```

                                0000 00000 MEMBER_NAME:
                                .WORD  Save nothing
                                CLRL  -(SP)
                                PUSHAB 28(TPARSE_BLOCK)
                                PUSHAB 16(TPARSE_BLOCK)
00000000G 00                   03 FB 0000A  CALLS  #3, SYS$ASCTOID
                                50 E9 00011  BLBC  STATUS, 1$
                                50 D4 00014  CLRL  R0
                                1E AC 26 AC B1 00016  CMPW  38(TPARSE_BLOCK), 30(TPARSE_BLOCK)
                                02 12 0001B  BNEQ  1$
                                50 D6 0001D  INCL  R0
                                04 0001F 1$:  RET
: 1113
: 1121
: 1122
: 1124
: 1126

```

; Routine Size: 32 bytes, Routine Base: _LIB\$CODE + 05FB

```
1135 1127 1 |  
1136 1128 1 | State table to parse UIC string.  
1137 1129 1 |  
1138 1130 1 |  
1139 1131 1 $INIT_STATE (LIB$$UIC_STATE_TABLE, LIB$$UIC_KEY_TABLE, _LIB$PARSE);  
1140 1132 1 |  
1141 P 1133 1 $STATE (  
1142 P 1134 1 ('[',,SET_BRACKET),  
1143 P 1135 1 ('<',,SET_BRACKET),  
1144 P 1136 1 ('%' ,HEX_FORMAT)  
1145 1137 1 );  
1146 1138 1 |  
1147 P 1139 1 $STATE (  
1148 P 1140 1 ('*' ,WILD_GROUP),  
1149 P 1141 1 (TPAS_OCTAL,,STORE_GROUP)  
1150 1142 1 );  
1151 1143 1 |  
1152 P 1144 1 $STATE (  
1153 P 1145 1 (' ,')  
1154 1146 1 );  
1155 1147 1 |  
1156 P 1148 1 $STATE (  
1157 P 1149 1 ('*' ,WILD_MEMBER),  
1158 P 1150 1 (TPAS_OCTAL,,STORE_MEMBER)  
1159 1151 1 );  
1160 1152 1 |  
1161 P 1153 1 $STATE (  
1162 P 1154 1 (']' ,TPAS_EXIT,CHECK_BRACKET),  
1163 P 1155 1 ('>' ,TPAS_EXIT,CHECK_BRACKET)  
1164 1156 1 );  
1165 1157 1 |  
1166 P 1158 1 $STATE (HEX_FORMAT,  
1167 P 1159 1 ('X')  
1168 1160 1 );  
1169 1161 1 |  
1170 P 1162 1 $STATE (  
1171 P 1163 1 (TPAS_HEX,TPAS_EXIT)  
1172 1164 1 );  
1173 1165 1 |  
1174 1166 1 |  
1175 1167 1 | State table to parse UIC or identifier string.  
1176 1168 1 |  
1177 1169 1 |  
1178 1170 1 $INIT_STATE (LIB$$IDENT_STATE_TABLE, LIB$$IDENT_KEY_TABLE, _LIB$PARSE);  
1179 1171 1 |  
1180 P 1172 1 $STATE (  
1181 P 1173 1 ('*' ,TPAS_EXIT,ALL_WILD),  
1182 P 1174 1 (TPAS_SYMBOL,TPAS_EXIT,SINGLE_NAME),  
1183 P 1175 1 ('[' ,SET_BRACKET),  
1184 P 1176 1 ('<' ,SET_BRACKET),  
1185 P 1177 1 ('%' ,HEX_FORMAT)  
1186 1178 1 );  
1187 1179 1 |  
1188 P 1180 1 $STATE (  
1189 P 1181 1 ((GROUP_MEMBER)),  
1190 P 1182 1 ('*' ,ALL_WILD),  
1191 P 1183 1 (TPAS_SYMBOL,,SINGLE_NAME)
```

```

: 1192          1184 1      );
: 1193          1185 1
: 1194          P 1186 1  $STATE (
: 1195          P 1187 1      (']',TPAS_EXIT,CHECK_BRACKET),
: 1196          P 1188 1      ('>',TPAS_EXIT,CHECK_BRACKET)
: 1197          1189 1      );
: 1198          1190 1
: 1199          P 1191 1  $STATE (GROUP_MEMBER,
: 1200          P 1192 1      ('*',WILD_GROUP),
: 1201          P 1193 1      (TPAS_OCTAL,,STORE_GROUP),
: 1202          P 1194 1      (TPAS_SYMBOL,,GROUP_NAME)
: 1203          1195 1      );
: 1204          1196 1
: 1205          P 1197 1  $STATE (
: 1206          P 1198 1      (',')
: 1207          1199 1      );
: 1208          1200 1
: 1209          P 1201 1  $STATE (
: 1210          P 1202 1      ('*',TPAS_EXIT,WILD_MEMBER),
: 1211          P 1203 1      (TPAS_OCTAL,TPAS_EXIT,STORE_MEMBER),
: 1212          P 1204 1      (TPAS_SYMBOL,TPAS_EXIT,MEMBER_NAME)
: 1213          1205 1      );
: 1214          1206 1
: 1215          1207 1  END
: 1216          1208 0  ELUDOM

```

.PSECT _LIBSPARSE_STATE,NOWRT, SHR, PIC,1

```

00000 LIB$$UIC_STATE_TABLE::
      805B 00000 ;TPATYPE      .BLKB 0
00000000* 00002 ;TPASACTION  U.2: .WORD -32677
      803C 00006 ;TPATYPE      U.3: .LONG <<SET_BRACKET-U.3>-4>
00000000* 00008 ;TPASACTION  U.4: .WORD -32708
      1425 0000C ;TPATYPE      U.5: .LONG <<SET_BRACKET-U.5>-4>
0000* 0000E ;TPASTARGET  U.6: .WORD 5157
      802A 00010 ;TPATYPE      U.7: .WORD <<U.7-U.8>-2>
00000000* 00012 ;TPASACTION  U.8: .WORD -32726
      85F4 00016 ;TPATYPE      U.9: .WORD <<WILD_GROUP-U.10>-4>
00000000* 00018 ;TPASACTION  U.10: .LONG <<STORE_GROUP-U.12>-4>
      042C 0001C ;TPATYPE      U.11: .WORD -31244
      802A 0001E ;TPATYPE      U.12: .LONG <<STORE_GROUP-U.12>-4>
00000000* 00020 ;TPASACTION  U.13: .WORD 1068
      U.14: .WORD -32726

```

85F4	00024	U.15:	LONG	<<WILD_MEMBER-U.15>-4>	:
		:TPASTYPE			:
00000000*	00026	U.16:	WORD	-31244	:
		:TPASACTION			:
905D	0002A	U.17:	LONG	<<STORE_MEMBER-U.17>-4>	:
		:TPASTYPE			:
00000000*	0002C	U.18:	WORD	-28579	:
		:TPASACTION			:
FFFF	00030	U.19:	LONG	<<CHECK_BRACKET-U.19>-4>	:
		:TPASTARGET			:
943E	00032	U.20:	WORD	-1	:
		:TPASTYPE			:
00000000*	00034	U.21:	WORD	-27536	:
		:TPASACTION			:
FFFF	00038	U.22:	LONG	<<CHECK_BRACKET-U.22>-4>	:
		:TPASTARGET			:
	0003A	U.23:	WORD	-1	:
		:HEX_FORMAT			:
0458	0003A	U.7:	BLKB	0	:
		:TPASTYPE			:
15F5	0003C	U.24:	WORD	1112	:
		:TPASTYPE			:
FFFF	0003E	U.25:	WORD	5621	:
		:TPASTARGET			:
	00040	U.26:	WORD	-1	:
		LIB\$IDENT STATE TABLE::			:
902A	00040	:TPASTYPE			:
		BLKB	0		:
00000000*	00042	U.28:	WORD	-28630	:
		:TPASACTION			:
FFFF	00046	U.29:	LONG	<<ALL_WILD-U.29>-4>	:
		:TPASTARGET			:
91F1	00048	U.30:	WORD	-1	:
		:TPASTYPE			:
00000000*	0004A	U.31:	WORD	-28175	:
		:TPASACTION			:
FFFF	0004E	U.32:	LONG	<<SINGLE_NAME-U.32>-4>	:
		:TPASTARGET			:
805B	00050	U.33:	WORD	-1	:
		:TPASTYPE			:
00000000*	00052	U.34:	WORD	-32677	:
		:TPASACTION			:
803C	00056	U.35:	LONG	<<SET_BRACKET-U.35>-4>	:
		:TPASTYPE			:
00000000*	00058	U.36:	WORD	-32708	:
		:TPASACTION			:
1425	0005C	U.37:	LONG	<<SET_BRACKET-U.37>-4>	:
		:TPASTYPE			:
0000*	0005E	U.38:	WORD	5157	:
		:TPASTARGET			:
09F8	00060	U.39:	WORD	<<U.7-U.39>-2>	:
		:TPASTYPE			:
0000*	00062	U.40:	WORD	2552	:
		:TPASSUBEXP			:
802A	00064	U.42:	WORD	<<U.41-U.42>-2>	:
		:TPASTYPE			:
		U.43:	WORD	-32726	:


```
00000000* 00066 ;TP$ACTION
                U.44: .LONG    <<ALL_WILD-U.44>-4>
                ;
                85F1 0006A ;TP$TYPE
                U.45: .WORD    -31247
                ;
00000000* 0006C ;TP$ACTION
                U.46: .LONG    <<SINGLE_NAME-U.46>-4>
                ;
                905D 00070 ;TP$TYPE
                U.47: .WORD    -28579
                ;
00000000* 00072 ;TP$ACTION
                U.48: .LONG    <<CHECK_BRACKET-U.48>-4>
                ;
                FFFF 00076 ;TP$TARGET
                U.49: .WORD    -1
                ;
                943E 00078 ;TP$TYPE
                U.50: .WORD    -27586
                ;
00000000* 0007A ;TP$ACTION
                U.51: .LONG    <<CHECK_BRACKET-U.51>-4>
                ;
                FFFF 0007E ;TP$TARGET
                U.52: .WORD    -1
                ;
                00080 ;GROUP_MEMBER
                U.41: .BLKB    0
                ;
                802A 00080 ;TP$TYPE
                U.53: .WORD    -32726
                ;
00000000* 00082 ;TP$ACTION
                U.54: .LONG    <<WILD_GROUP-U.54>-4>
                ;
                81F4 00086 ;TP$TYPE
                U.55: .WORD    -32268
                ;
00000000* 00088 ;TP$ACTION
                U.56: .LONG    <<STORE_GROUP-U.56>-4>
                ;
                85F1 0008C ;TP$TYPE
                U.57: .WORD    -31247
                ;
00000000* 0008E ;TP$ACTION
                U.58: .LONG    <<GROUP_NAME-U.58>-4>
                ;
                042C 00092 ;TP$TYPE
                U.59: .WORD    1068
                ;
                902A 00094 ;TP$TYPE
                U.60: .WORD    -28630
                ;
00000000* 00096 ;TP$ACTION
                U.61: .LONG    <<WILD_MEMBER-U.61>-4>
                ;
                FFFF 0009A ;TP$TARGET
                U.62: .WORD    -1
                ;
                91F4 0009C ;TP$TYPE
                U.63: .WORD    -28172
                ;
00000000* 0009E ;TP$ACTION
                U.64: .LONG    <<STORE_MEMBER-U.64>-4>
                ;
                FFFF 000A2 ;TP$TARGET
                U.65: .WORD    -1
                ;
                9577 000A4 ;TP$TYPE
                U.66: .WORD    -27151
                ;
00000000* 000A6 ;TP$ACTION
                U.67: .LONG    <<MEMBER_NAME-U.67>-4>
                ;
                FFFF 000AA ;TP$TARGET
                U.68: .WORD    -1
                ;
```

.PSECT _LIB\$PARSE_KEY0, NOWRT, SHR, PIC, 1

00000 LIB\$SUIC_KEY TABLE::
 .BLKB 0

Mac
_S2
604
The
MAC

00000 ;TPASKEY0
U.1: .BLKB 0
00000 LIB\$\$IDENT_KEY_TABLE::
BCKB 0
00000 ;TPASKEY0
U.27: .BLKB 0

PSECT SUMMARY

Name	Bytes	Attributes
LIB\$CODE	1563	NOVEC,NOWRT, RD ; EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
LIB\$PARSE_KEY0	0	NOVEC,NOWRT, RD ; EXE, SHR, LCL, REL, CON, PIC,ALIGN(1)
LIB\$PARSE_STATE	172	NOVEC,NOWRT, RD ; EXE, SHR, LCL, REL, CON, PIC,ALIGN(1)

Library Statistics

File	Symbols		Pages Mapped	Processing Time
	Total	Loaded Percent		
\$255\$DUA28:[SYSLIB]STARLET.L32;1	9776	28 0	581	00:01.0
\$255\$DUA28:[SYSLIB]TPAMAC.L32;1	42	33 78	14	00:00.1

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/NOTRACE/LIS=LIS\$:TPARSE/OBJ=OBJ\$:TPARSE MSRCS:TPARSE/UPDATE=(ENH\$:TPARSE)

: Size: 1531 code + 204 data bytes
: Run Time: 00:41.3
: Elapsed Time: 01:26.4
: Lines/CPU Min: 1753
: Lexemes/CPU-Min: 48464
: Memory Used: 341 pages
: Compilation Complete

LIBSPAWN
LIS

LIBSTATUM
LIS

LIBTRAAZE
LIS

LIBSPANC
LIS

LIBSYMBOL
LIS

LIBTRNLOG
LIS

LIBSKPC
LIS

LIBTIMER
LIS

LIBTPARSE
LIS

LIBTRIMF1
LIS

LIBSTRET
LIS

LIBTRAE2A
LIS