


```
DDDDDDDD  CCCCCCCC  LL  DDDDDDDD  IIIIII  EEEEEEEEE  TTTTTTTTTT
DDDDDDDD  CCCCCCCC  LL  DDDDDDDD  IIIIII  EEEEEEEEE  TTTTTTTTTT
DD  DD  CC  DD  DD  LL  DD  DD  EE  TT
DD  DD  CC  DD  DD  LL  DD  DD  EE  TT
DD  DD  CC  DD  DD  LL  DD  DD  EE  TT
DD  DD  CC  DD  DD  LL  DD  DD  EE  TT
DD  DD  CC  DD  DD  LL  DD  DD  EE  TT
DD  DD  CC  DD  DD  LL  DD  DD  EE  TT
DD  DD  CC  DD  DD  LL  DD  DD  EE  TT
DD  DD  CC  DD  DD  LL  DD  DD  EE  TT
DDDDDDDD  CCCCCCCC  LLLLLLLLLL  DDDDDDDD  IIIIII  EEEEEEEEE  TTTTTTTTTT
DDDDDDDD  CCCCCCCC  LLLLLLLLLL  DDDDDDDD  IIIIII  EEEEEEEEE  TTTTTTTTTT
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
LLLLLLLLLL  IIIIII  SSSSSSSS
LLLLLLLLLL  IIIIII  SSSSSSSS
```



```
0001 0 %title 'DCLDIET - Put a DCL Procedure on a Diet'
0002 0 MODULE dcldiet (main=dcldiet$main, ident = 'V04-000') =
0003 1 BEGIN
0004 1
0005 1 *****
0006 1 *
0007 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0008 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0009 1 * ALL RIGHTS RESERVED.
0010 1 *
0011 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0012 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0013 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0014 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0015 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0016 1 * TRANSFERRED.
0017 1 *
0018 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0019 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0020 1 * CORPORATION.
0021 1 *
0022 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0023 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0024 1 *
0025 1 *
0026 1 *****
0027 1
0028 1 ++
0029 1 Facility: DCLDIET, Put a DCL Procedure on a Diet
0030 1
0031 1 Abstract: The purpose of this program is to reduce the size of a DCL
0032 1 command procedure. This is done principally by removing
0033 1 all comments. We also eliminate lines that only contain a
0034 1 dollar sign, and reduce all whitespace to a single blank.
0035 1
0036 1 By shipping command procedures in this reduced format, we
0037 1 save space and speed them up.
0038 1
0039 1 Restrictions: The following restrictions must be obeyed in order that this
0040 1 program produce a correct command procedure.
0041 1
0042 1 1. Quoted strings must have closing quotation marks.
0043 1
0044 1 2. Continuation hypens and comment exclamation points may
0045 1 not be lexically substituted into or out of a line.
0046 1
0047 1 3. The DECK and EOD commands may not be abbreviated.
0048 1
0049 1 Environment: Native, User mode.
0050 1
0051 1 Author: Paul C. Anagnostopoulos, Creation Date: 7 September 1982
0052 1
0053 1 Revision History:
0054 1
0055 1 V03-001 DAS0001 David Solomon 30-Apr-1984
0056 1 Fix two accvios resulting from reading zero-length records.
0057 1 --
```

DCLDIET
V04-000

DCLDIET - Put a DCL Procedure on a Diet

E 6
16-Sep-1984 02:18:55
14-Sep-1984 13:25:20

VAX-11 Bliss-32 V4.0-742
[UTIL32.SRC]DCLDIET.B32;1

Page 2
(1)

: 58
: 59
: 60

0058 1
0059 1
0060 1 library 'sys\$library:starlet';

DCL
V04


```
62 0061 1 %sbttl 'Module Definitions'
63 0062 1
64 0063 1 ! Macro Definitions:
65 0064 1
66 0065 1 ! The following definitions are used to create extensions to the Bliss
67 0066 1 ! language. This makes writing programs easier, and hopefully also
68 0067 1 ! facilitates reading them.
69 0068 1
70 0069 1 ! We begin with the definition of a boolean data type.
71 0070 1
72 M 0071 1 macro boolean =
73 M 0072 1     byte
74 0073 1 %;
75 0074 1
76 0075 1 literal
77 0076 1     false = 0,
78 0077 1     true  = 1;
79 0078 1
80 0079 1 ! Let's define symbolic names for various control characters that we use
81 0080 1 ! a lot.
82 0081 1
83 0082 1 literal
84 0083 1     HT      = %x'09';
85 0084 1
86 0085 1 ! Now let's define a descriptor data type. This allows us to declare
87 0086 1 ! variables, particularly parameters, as descriptors.
88 0087 1
89 0088 1 field descriptor_fields = set
90 0089 1     len      = [dsc$w_length],
91 0090 1     typ      = [dsc$b_dtype],
92 0091 1     cls      = [dsc$b_class],
93 0092 1     ptr      = [dsc$a_pointer]
94 0093 1 tes;
95 0094 1
96 M 0095 1 macro descriptor =
97 M 0096 1     block[8,byte] field(descriptor_fields)
98 0097 1 %;
99 0098 1
100 0099 1 ! Now we need some macros to better deal with strings. We will begin with
101 0100 1 ! three that can be used to generate string constants.
102 0101 1
103 M 0102 1 macro text(string) =
104 M 0103 1     uplit byte(string)
105 0104 1 %,
106 M 0105 1     ctext(string) =
107 M 0106 1     uplit byte(%ascic string)
108 0107 1 %,
109 M 0108 1     dtext(string) =
110 M 0109 1     %ascid string
111 0110 1 %;
112 0111 1
113 0112 1 ! Now we will define a macro that generates a buffer with accompanying
114 0113 1 ! descriptor. The user can specify the name and length of the buffer.
115 0114 1 ! This macro can appear in global or own declarations.
116 0115 1
117 M 0116 1 macro dbuffer(name,length) =
118 M 0117 1     name: block[8+length,byte] field(descriptor_fields) preset(
```

```
: 119      M 0118 1      [len] = length,  
: 120      M 0119 1      [typ] = 0,  
: 121      M 0120 1      [cls] = 0,  
: 122      M 0121 1      [ptr] = name + 8  
: 123      M 0122 1      )  
: 124      M 0123 1      %;  
: 125      M 0124 1  
: 126      M 0125 1      ! We also want a macro which helps us fill in a descriptor dynamically.  
: 127      M 0126 1      ! It lets us specify the descriptor, and the length and address to fill in.  
: 128      M 0127 1  
: 129      M 0128 1      macro build_descriptor(name,length,address) =  
: 130      M 0129 1          (name[len] = length;  
: 131      M 0130 1          name[typ] = 0;  
: 132      M 0131 1          name[cls] = 0;  
: 133      M 0132 1          name[ptr] = address;)  
: 134      M 0133 1      %;  
: 135      M 0134 1  
: 136      M 0135 1      ! Now we are going to define some constructs that let us do common sorts  
: 137      M 0136 1      ! of operations more elegantly. They make things more meaningful and  
: 138      M 0137 1      ! are also faster to write.  
: 139      M 0138 1  
: 140      M 0139 1      ! Begin with two macros to increment and decrement a variable.  
: 141      M 0140 1  
: 142      M 0141 1      macro increment(variable) =  
: 143      M 0142 1          (variable = .variable + 1)  
: 144      M 0143 1      %,  
: 145      M 0144 1          decrement(variable) =  
: 146      M 0145 1          (variable = .variable - 1)  
: 147      M 0146 1      %;  
: 148      M 0147 1  
: 149      M 0148 1      ! Define a macro that can check a status. If not successful, then it will  
: 150      M 0149 1      ! signal the remaining arguments.  
: 151      M 0150 1  
: 152      M 0151 1      macro check(status) =  
: 153      M 0152 1          (if not status then  
: 154      M 0153 1              signal(%remaining);)  
: 155      M 0154 1      %;
```



```
157 0155 1 ! Table of Contents:
158 0156 1
159 0157 1 forward routine
160 0158 1     dcldiet$main: novalue,
161 0159 1     dcldiet$reduce: novalue,
162 0160 1     dcldiet$scrunch_line: novalue,
163 0161 1     dcldiet$skip_whitespace: novalue;
164 0162 1
165 0163 1 ! External References:
166 0164 1
167 0165 1 external routine
168 0166 1     cli$get_value: addressing_mode(general);
169 0167 1
170 0168 1 ! Shared Messages:
171 0169 1
172 P 0170 1 $shr_msgdef(dcldiet, 3, local,
173 P 0171 1     (closein, warning),
174 P 0172 1     (closeout, warning),
175 P 0173 1     (openin, severe),
176 P 0174 1     (openout, severe),
177 P 0175 1     (readerr, error),
178 0176 1     (writeerr, error));
179 0177 1
180 0178 1 ! Own Variables:
181 0179 1
182 0180 1 ! We need a set of RMS control blocks to open and read the input file.
183 0181 1
184 0182 1 own
185 0183 1     dbuffer(input_esa,nam$c_maxrss),
186 0184 1     dbuffer(input_rsa,nam$c_maxrss),
187 0185 1
188 P 0186 1     input_nam: $nam(esa=input_esa+8,
189 P 0187 1         ess=nam$c_maxrss,
190 P 0188 1         rsa=input_rsa+8,
191 0189 1         rss=nam$c_maxrss),
192 0190 1
193 P 0191 1     input_fab: $fab(dnm='.com',
194 P 0192 1         fac=get,
195 P 0193 1         nam=input_nam,
196 0194 1         shr=get),
197 0195 1
198 0196 1     input_ubf: block[1024,byte],
199 0197 1
200 P 0198 1     input_rab: $rab(fab=input_fab,
201 P 0199 1         mbc=4,
202 P 0200 1         mbf=2,
203 P 0201 1         rac=seq,
204 P 0202 1         ubf=input_ubf,
205 0203 1         usz=1024);
206 0204 1
207 0205 1 ! We need a set of RMS control blocks to create the output file.
208 0206 1
209 0207 1 own
210 0208 1     dbuffer(output_esa,nam$c_maxrss),
211 0209 1
212 P 0210 1     output_nam: $nam(rlf=input_nam,
213 P 0211 1         esa=output_esa+8,
```



```
: 214      0212 1      ess=nam$c_maxrss),
: 215      0213 1
: 216      P 0214 1      output_fab: $fab(fac=put,
: 217      P 0215 1      fop=ofp,
: 218      P 0216 1      mrs=1024,
: 219      P 0217 1      nam=output_nam,
: 220      P 0218 1      org=seq,
: 221      P 0219 1      rat=cr,
: 222      0220 1      rfm=var),
: 223      0221 1
: 224      0222 1      output_ubf: block[1024,byte],
: 225      0223 1
: 226      P 0224 1      output_rab: $rab(fab=output_fab,
: 227      P 0225 1      mbc=4,
: 228      P 0226 1      mbf=2,
: 229      0227 1      rac=seq);
```



```
231 0228 1 %sbttl 'DCLDIET$MAIN - Main Routine'
232 0229 1 ++
233 0230 1 Functional Description:
234 0231 1 This is the main routine for the DCLDIET command. It parses the
235 0232 1 command line and then opens up the input file.
236 0233 1
237 0234 1 Formal Parameters:
238 0235 1 none
239 0236 1
240 0237 1 Returned Value:
241 0238 1 none
242 0239 1
243 0240 1 Notes:
244 0241 1 none
245 0242 1 --
246 0243 1
247 0244 1
248 0245 2 GLOBAL ROUTINE dclldiet$main: novalue = BEGIN
249 0246 2
250 0247 2 local
251 0248 2 status: long,
252 0249 2 work_dsc: descriptor;
253 0250 2
254 0251 2
255 0252 2 ! Begin by obtaining the name of the input file and opening it.
256 0253 2
257 0254 2 status = cli$get_value(dtext('P1'),input_esa);
258 0255 2 input_fab[fab$l_fna] = .input_esa[ptr];
259 0256 2 input_fab[fab$b_fns] = .input_esa[len];
260 0257 2 status = $open(fab=input_fab);
261 0258 2 dclldiet$scrunch_line(input_esa);
262 0259 2 check(.status, dclldiet$_openin,1,input_esa,.status,.input_fab[fab$l_stv]);
263 0260 2 status = $connect(rab=input_rab);
264 0261 2 check(.status, dclldiet$_openin,1,input_esa,.status,.input_rab[rab$l_stv]);
265 0262 2
266 0263 2 ! Now we can open the output file. The optional /OUTPUT qualifier will
267 0264 2 ! tell us the name of the output file.
268 0265 2
269 0266 2 status = cli$get_value(dtext('OUTPUT'),output_esa);
270 0267 2 output_fab[fab$l_fna] = .output_esa[ptr];
271 0268 2 output_fab[fab$b_fns] = .output_esa[len];
272 0269 2 status = $create(fab=output_fab);
273 0270 2 dclldiet$scrunch_line(output_esa);
274 0271 2 check(.status, dclldiet$_openout,1,output_esa,.status,.output_fab[fab$l_stv]);
275 0272 2 status = $connect(rab=output_rab);
276 0273 2 check(.status, dclldiet$_openout,1,output_esa,.status,.output_rab[rab$l_stv]);
277 0274 2
278 0275 2 ! Call a routine to process the input command procedure and produce the output.
279 0276 2
280 0277 2 dclldiet$reduce();
281 0278 2
282 0279 2 ! Close the input and output files.
283 0280 2
284 0281 2 status = $close(fab=input_fab);
285 0282 2 check(.status, dclldiet$_closein,1,input_esa,.status,.input_fab[fab$l_stv]);
286 0283 2 status = $close(fab=output_fab);
287 0284 2 check(.status, dclldiet$_closeout,1,output_esa,.status,.output_fab[fab$l_stv]);
```


DCLDIET
V04-000

DCLDIET - Put a DCL Procedure on a Diet
DCLDIET\$MAIN - Main Routine

K 6
16-Sep-1984 02:18:55
14-Sep-1984 13:25:20

VAX-11 Bliss-32 V4.0-742
[UTIL32.SRC]DCLDIET.B32;1

Page 8
(4)

: 288
: 289
: 290

0285 2
0286 2 return;
0287 1 END;

.TITLE DCLDIET DCLDIET - Put a DCL Procedure on a Diet
.IDENT \V04-000\

.PSECT \$SPLITS, NOWRT, NOEXE, 2

6D	6F	63	2E	00000	P.AAA:	.ASCII	\.com\	:
00	00	31	50	00004	P.AAC:	.ASCII	\P1\<0><0>	:
		010E0002		00008	P.AAB:	.LONG	17694722	:
		00000000		0000C		.ADDRESS	P.AAC	:
00	00	54	55	50	54	55	4F	:
		010E0006		00010	P.AAE:	.ASCII	\OUTPUT\<0><0>	:
		00000000		00018	P.AAD:	.LONG	17694726	:
				0001C		.ADDRESS	P.AAE	:

.PSECT \$OWNS, NOEXE, 2

00FF	00000	INPUT_ESA:		
		.WORD	255	:
00	00	.BYTE	0, 0	:
00000000	00002	.ADDRESS	INPUT_ESA+8	:
	00004	.BLKB	255	:
	00107	.BLKB	1	:
00FF	00108	INPUT_RSA:		:
		.WORD	255	:
00	00	.BYTE	0, 0	:
00000000	0010A	.ADDRESS	INPUT_RSA+8	:
	0010C	.BLKB	255	:
	00110	.BLKB	1	:
	0020F			:
02	00210	INPUT_NAM:		:
		.BYTE	2	:
60	00211	.BYTE	96	:
FF	00212	.BYTE	-1	:
00	00213	.BYTE	0	:
00000000	00214	.ADDRESS	INPUT_RSA+8	:
00	00218	.BYTE	0	:
00	00219	.BYTE	0	:
FF	0021A	.BYTE	-1	:
00	0021B	.BYTE	0	:
00000000	0021C	.ADDRESS	INPUT_ESA+8	:
00000000	00220	.LONG	0	:
0000#	00224	.WORD	0[8]	:
0000#	00234	.WORD	0[3]	:
0000#	0023A	.WORD	0[3]	:
00000000	00240	.LONG	0	:
00000000	00244	.LONG	0	:
00	00248	.BYTE	0	:
00	00249	.BYTE	0	:
00	0024A	.BYTE	0	:
00	0024B	.BYTE	0	:
00	0024C	.BYTE	0	:
00	0024D	.BYTE	0	:
00#	0024E	.BYTE	0[2]	:
00000000	00250	.LONG	0	:


```
00000000 00254 .LONG 0
00000000 00258 .LONG 0
00000000 0025C .LONG 0
00000000 00260 .LONG 0
00000000 00264 .LONG 0
00000000# 00268 .LONG 0[2]
03 00270 INPUT_FAB:
      50 00271 .BYTE 3
      0000 00272 .WORD 80
00000000 00274 .LONG 0
00000000 00278 .LONG 0
00000000 0027C .LONG 0
00000000 00280 .LONG 0
      0000 00284 .WORD 0
      02 00286 .BYTE 2
      02 00287 .BYTE 2
00000000 00288 .LONG 0
      00 0028C .BYTE 0
      00 0028D .BYTE 0
      00 0028E .BYTE 0
      02 0028F .BYTE 2
00000000 00290 .LONG 0
00000000 00294 .LONG 0
00000000 00298 .ADDRESS INPUT_NAM
00000000 0029C .LONG 0
00000000 002A0 .ADDRESS P.AAA
      00 002A4 .BYTE 0
      04 002A5 .BYTE 4
      0000 002A6 .WORD 0
00000000 002A8 .LONG 0
      0000 002AC .WORD 0
      00 002AE .BYTE 0
      00 002AF .BYTE 0
00000000 002B0 .LONG 0
00000000 002B4 .LONG 0
      0000 002B8 .WORD 0
      00 002BA .BYTE 0
      00 002BB .BYTE 0
00000000 002BC .LONG 0
      002C0 INPUT_UBF:
      01 006C0 INPUT_RAB:
      44 006C1 .BYTE 1
      0000 006C2 .WORD 68
00000000 006C4 .LONG 0
00000000 006C8 .LONG 0
00000000 006CC .LONG 0
      0000# 006D0 .WORD 0[3]
      0000 006D6 .WORD 0
00000000 006D8 .LONG 0
      0000 006DC .WORD 0
      00 006DE .BYTE 0
      00 006DF .BYTE 0
      0400 006E0 .WORD 1024
      0000 006E2 .WORD 0
```



```
00000000' 006E4      .ADDRESS INPUT_UBF
00000000 006E8      .LONG 0
00000000 006EC      .LONG 0
00000000 006F0      .LONG 0
      00 006F4      .BYTE 0
      00 006F5      .BYTE 0
      02 006F6      .BYTE 2
      04 006F7      .BYTE 4
00000000 006F8      .LONG 0
00000000' 006FC      .ADDRESS INPUT_FAB
00000000 00700      .LONG 0
      00FF 00704 OUTPUT_ESA:
      00 00 00706      .WORD 255
00000000' 00708      .BYTE 0, 0
      0070C      .ADDRESS OUTPUT_ESA+8
      0080B      .BLKB 255
      02 0080C OUTPUT_NAM:
      60 0080D      .BLKB 1
      00 0080E      .BYTE 2
      00 0080F      .BYTE 96
00000000 00810      .BYTE 0
      00 00814      .LONG 0
      00 00815      .BYTE 0
      FF 00816      .BYTE 0
      00 00817      .BYTE -1
00000000' 00818      .BYTE 0
00000000' 0081C      .ADDRESS OUTPUT_ESA+8
      0000# 00820      .ADDRESS INPUT_NAM
      0000# 00830      .WORD 0[8]
      0000# 00836      .WORD 0[3]
00000000 0083C      .WORD 0[3]
00000000 00840      .LONG 0
      00 00844      .LONG 0
      00 00845      .BYTE 0
      00 00846      .BYTE 0
      00 00847      .BYTE 0
      00 00848      .BYTE 0
      00 00849      .BYTE 0
      00# 0084A      .BYTE 0[2]
00000000 0084C      .LONG 0
00000000 00850      .LONG 0
00000000 00854      .LONG 0
00000000 00858      .LONG 0
00000000 0085C      .LONG 0
00000000 00860      .LONG 0
00000000# 00864      .LONG 0[2]
      03 0086C OUTPUT_FAB:
      50 0086D      .BYTE 3
      0000 0086E      .BYTE 80
20000000 00870      .WORD 0
00000000 00874      .LONG 536870912
00000000 00878      .LONG 0
00000000 0087C      .LONG 0
      0000 00880      .LONG 0
      .WORD 0
```



```
01 00882 .BYTE 1
00 00883 .BYTE 0
00000000 00884 .LONG 0
00 00888 .BYTE 0
00 00889 .BYTE 0
02 0088A .BYTE 2
02 0088B .BYTE 2
00000000 0088C .LONG 0
00000000 00890 .LONG 0
00000000 00894 .ADDRESS OUTPUT_NAM
00000000 00898 .LONG 0
00000000 0089C .LONG 0
00 008A0 .BYTE 0
00 008A1 .BYTE 0
0400 008A2 .WORD 1024
00000000 008A4 .LONG 0
0000 008A8 .WORD 0
00 008AA .BYTE 0
00 008AB .BYTE 0
00000000 008AC .LONG 0
00000000 008B0 .LONG 0
0000 008B4 .WORD 0
00 008B6 .BYTE 0
00 008B7 .BYTE 0
00000000 008B8 .LONG 0
008BC OUTPUT_UBF:
01 00CBC OUTPUT_RAB:
44 00CBD .BYTE 1
0000 00CBE .BYTE 68
00000000 00CC0 .WORD 0
00000000 00CC4 .LONG 0
00000000 00CC8 .LONG 0
0000# 00CCC .WORD 0[3]
0000 00CD2 .WORD 0
00000000 00CD4 .LONG 0
0000 00CD8 .WORD 0
00 00CDA .BYTE 0
00 00CDB .BYTE 0
0000 00CDC .WORD 0
0000 00CDE .WORD 0
00000000 00CE0 .LONG 0
00000000 00CE4 .LONG 0
00000000 00CE8 .LONG 0
00000000 00CEC .LONG 0
00 00CF0 .BYTE 0
00 00CF1 .BYTE 0
02 00CF2 .BYTE 2
04 00CF3 .BYTE 4
00000000 00CF4 .LONG 0
00000000 00CF8 .ADDRESS OUTPUT_FAB
00000000 00CFC .LONG 0

.EXTRN CLISGET VALUE, SYSSOPEN
.EXTRN SYSSCONNECT, SYSSCREATE
.EXTRN SYSSCLOSE
```



```
.PSECT $CODE$,NOWRT,2

.ENTRY DCLDIET$MAIN, Save R2,R3,R4,R5,R6,R7      0245
MOVAB SY$$CLOSE, R7
MOVAB SY$$CONNECT, R6
MOVAB CLISGET_VALUE, R5
MOVAB LIB$$SIGNAL, R4
MOVAB INPUT_ESA, R3
SUBL2 #8, SP
PUSHL R3      0254
PUSHAB P.AAB
CALLS #2, CLISGET_VALUE
MOVL R0, STATUS
MOVL INPUT_ESA+4, INPUT_FAB+44      0255
MOVB INPUT_ESA, INPUT_FAB+52      0256
PUSHAB INPUT_FAB      0257
CALLS #1, SY$$OPEN
MOVL R0, STATUS
PUSHL R3      0258
CALLS #1, DCLDIET$$SCRUNCH_LINE
BLBS STATUS, 1$      0259
PUSHL INPUT_FAB+12
PUSHL STATUS
PUSHL R3
PUSHL #1
PUSHL #200860
CALLS #5, LIB$$SIGNAL
PUSHAB INPUT_RAB      0260
CALLS #1, SY$$CONNECT
MOVL R0, STATUS
BLBS STATUS, 2$      0261
PUSHL INPUT_RAB+12
PUSHL STATUS
PUSHL R3
PUSHL #1
PUSHL #200860
CALLS #5, LIB$$SIGNAL
PUSHAB OUTPUT_ESA      0266
PUSHAB P.AAD
CALLS #2, CLISGET_VALUE
MOVL R0, STATUS
MOVL OUTPUT_ESA+4, OUTPUT_FAB+44      0267
MOVB OUTPUT_ESA, OUTPUT_FAB+52      0268
PUSHAB OUTPUT_FAB      0269
CALLS #1, SY$$CREATE
MOVL R0, STATUS
PUSHAB OUTPUT_ESA      0270
CALLS #1, DCLDIET$$SCRUNCH_LINE
BLBS STATUS, 3$      0271
PUSHL OUTPUT_FAB+12
PUSHL STATUS
PUSHAB OUTPUT_ESA
PUSHL #1
PUSHL #200868
CALLS #5, LIB$$SIGNAL
PUSHAB OUTPUT_RAB      0272
```


66	01	FB	000D7	CALLS	#1, SYSS\$CONNECT	:	
52	50	D0	000DA	MOVL	R0, STATUS	:	
15	52	E8	000DD	BLBS	STATUS, 4\$	: 0273	
	C3	DD	000E0	PUSHL	OUTPUT_FAB+12	:	
	52	DD	000E4	PUSHL	STATUS	:	
	C3	9F	000E6	PUSHAB	OUTPUT_ESA	:	
	01	DD	000EA	PUSHL	#1	:	
	8F	DD	000EC	PUSHL	#200868	:	
0000V	64	05	FB	000F2	CALLS	#5, LIB\$SIGNAL	:
CF	00	FB	000F5	CALLS	#0, DCLDIET\$REDUCE	: 0277	
	C3	9F	000FA	PUSHAB	INPUT_FAB	: 0281	
67	01	FB	000FE	CALLS	#1, SYSS\$CLOSE	:	
52	50	D0	00101	MOVL	R0, STATUS	:	
13	52	E8	00104	BLBS	STATUS, 5\$	: 0282	
	C3	DD	00107	PUSHL	INPUT_FAB+12	:	
	52	DD	0010B	PUSHL	STATUS	:	
	53	DD	0010D	PUSHL	R3	:	
	01	DD	0010F	PUSHL	#1	:	
	8F	DD	00111	PUSHL	#200784	:	
64	05	FB	00117	CALLS	#5, LIB\$SIGNAL	:	
	C3	9F	0011A	PUSHAB	OUTPUT_FAB	: 0283	
67	01	FB	0011E	CALLS	#1, SYSS\$CLOSE	:	
52	50	D0	00121	MOVL	R0, STATUS	:	
15	52	E8	00124	BLBS	STATUS, 6\$	: 0284	
	C3	DD	00127	PUSHL	OUTPUT_FAB+12	:	
	52	DD	0012B	PUSHL	STATUS	:	
	C3	9F	0012D	PUSHAB	OUTPUT_ESA	:	
	01	DD	00131	PUSHL	#1	:	
	8F	DD	00133	PUSHL	#200792	:	
64	05	FB	00139	CALLS	#5, LIB\$SIGNAL	:	
	04	0013C	6\$:	RET		: 0287	

; Routine Size: 317 bytes, Routine Base: \$CODE\$ + 0000

```
292 0288 1 %sbttl 'DCLDIET$REDUCE - Reduce a Command Procedure'
293 0289 1 ++
294 0290 1 Functional Description:
295 0291 1 This procedure is responsible for reducing the size of a command
296 0292 1 procedure by eliminating comments and compressing any whitespace.
297 0293 1 We read the entire input file and generate a new output file.
298 0294 1
299 0295 1 Formal Parameters:
300 0296 1 none
301 0297 1
302 0298 1 Returned Value:
303 0299 1 none
304 0300 1
305 0301 1 Notes:
306 0302 1 none
307 0303 1 --
308 0304 1
309 0305 1
310 0306 2 GLOBAL ROUTINE dcldiet$reduce : novalue = BEGIN
311 0307 2
312 0308 2 local
313 0309 2 status: long,
314 0310 2 line_dsc: descriptor,
315 0311 2 work_dsc: descriptor,
316 0312 2 continuation_line: boolean,
317 0313 2 within_deck: boolean,
318 0314 2 dcl_command: boolean;
319 0315 2
320 0316 2
321 0317 2 ! We will reduce the command procedure by reading each line, processing it,
322 0318 2 ! and perhaps writing a new line into the output file.
323 0319 2
324 0320 2 continuation_line = within_deck = false;
325 0321 2
326 0322 2 while true do (
327 0323 2
328 0324 2 ! Begin by getting the next line. We're done on end-of-file.
329 0325 2
330 0326 2 status = $get(rab=input_rab);
331 0327 2 if .status eqlu rms$_eof then exitloop;
332 0328 2 check(.status, dcldiet$_readerr, 1, input_esa, .status, .input_rab[rab$_l_stv]);
333 0329 2
334 0330 2 ! Set up the output RAB for writing the line. Also set up a working
335 0331 2 ! descriptor in case the line must be modified.
336 0332 2
337 0333 2 output_rab[rab$_w_rsz] = .input_rab[rab$_w_rsz];
338 0334 2 output_rab[rab$_l_rbf] = .input_rab[rab$_l_rbf];
339 0335 2 build_descriptor(work_dsc, .input_rab[rab$_w_rsz], .input_rab[rab$_l_rbf]);
340 0336 2
341 0337 2 ! Update the work descriptor to skip initial whitespace.
342 0338 2
343 0339 2 dcldiet$skip_whitespace(work_dsc);
344 0340 2
345 0341 2 ! Now we have to determine whether this is a DCL command line
346 0342 2 ! or a data line.
347 0343 2
348 0344 2 dcl_command =
```



```
349      (if not .within_deck and .work_dsc[len] gtru 0
350      then
351          (ch$rchar(.work_dsc[ptr]) eqlu '$')
352      else
353          false)
354      or .continuation_line;
355
356      ! If it's a DCL command line, we have to scrunch it. Skip to the
357      ! beginning of the verb, call the scrunching routine, and then put
358      ! back a dollar sign if necessary.
359
360      if .dcl_command then (
361          if not .continuation_line then (
362              decrement(work_dsc[len]);
363              increment(work_dsc[ptr]);
364          );
365          dcldiet$skip_whitespace(work_dsc);
366          dcldiet$scrunch_line(work_dsc);
367          if not .continuation_line then (
368              increment(work_dsc[len]);
369              decrement(work_dsc[ptr]);
370              ch$wchar('$',..work_dsc[ptr]);
371          );
372      );
373
374      ! Now we want to write the line into the output file. Don't bother
375      ! if it's a DCL command line with only a dollar sign.
376
377      if .work_dsc[len] gtru 1 or
378          .continuation_line or
379          not .dcl_command then (
380          if .dcl_command then (
381              output_rab[rab$w_rsz] = .work_dsc[len];
382              output_rab[rab$l_rbf] = .work_dsc[ptr];
383          );
384          status = $put(rab=output_rab);
385          check(.status, dcldiet$_writeerr,1,output_esa,.status,.output_rab[rab$l_stv]);
386      );
387
388      ! Next we determine the state of our two control flags.
389      ! The continuation flag is set if we just processed a DCL command
390      ! line and it has a hyphen at the end.
391
392      continuation_line =
393          (if .dcl_command
394          then
395              (ch$rchar(.work_dsc[ptr]+.work_dsc[len]-1) eqlu '-')
396          else false
397          );
398
399      ! The setting of the deck flag depends upon whether or not we just
400      ! processed a DCL command. If so, the flag is set if we had a DECK
401      ! command. If not, the flag is left alone unless we had an EOD command
402      ! while in a deck, in which case it is reset.
403
404      if .dcl_command then
405          within_deck = .work_dsc[len] eqlu 5 and
```

20

OFFC 00000				EXTRN	SYSS\$GET, SYSS\$PUT	
				.ENTRY	DCLDIET\$REDUCE, Save R2,R3,R4,R5,R6,R7,R8,-	0306
					R9,R10,R11	
				MOVAB	DCLDIET\$SKIP WHITESPACE, R11	
				MOVAB	LIB\$SIGNAL, R10	
				MOVAB	INPUT_RAB+34, R9	
				SUBL2	#16, SP	
				CLRB	WITHIN DECK	0320
				CLRB	CONTINUATION_LINE	
				PUSHAB	INPUT_RAB	0326
				CALLS	#1, SYSS\$GET	
				MOVL	R0, STATUS	
				CMPL	STATUS, #98938	0327
				BNEQ	2\$	
				RET		
				BLBS	STATUS, 3\$	0328
				PUSHL	INPUT_RAB+12	
				PUSHL	STATUS	
				PUSHAB	INPUT_ESA	
				PUSHL	#1	
				PUSHL	#200882	
				CALLS	#5, LIB\$SIGNAL	
				MOVW	INPUT_RAB+34, OUTPUT_RAB+34	0333
				MOVL	INPUT_RAB+40, OUTPUT_RAB+40	0334
				MOVZWL	INPUT_RAB+34, WORK_DSC	0335
				MOVL	INPUT_RAB+40, WORK_DSC+4	
				PUSHL	SP	0339
				CALLS	#1, DCLDIET\$SKIP_WHITESPACE	
				BLBS	WITHIN DECK, 4\$	0345
				TSTW	WORK_DSC	
				BEQL	4\$	
				CLRL	R0	0347

	24	04	BE 91 00069	CMPB	@WORK_DSC+4, #36		
			06 12 0006D	BNEQ	5\$		
			50 D6 0006F	INCL	R0		
			02 11 00071	BRB	5\$		
			50 D4 00073	CLRL	R0		0345
56	50		53 89 00075	BISB3	CONTINUATION_LINE, R0, DCL_COMMAND		0350
	20		56 E9 00079	BLBC	DCL_COMMAND, 7\$		0356
	05		53 E8 0007C	BLBS	CONTINUATION_LINE, 6\$		0357
			6E B7 0007F	DECW	WORK_DSC		0358
		04	AE D6 00081	INCL	WORK_DSC+4		0359
			5E DD 00084	PUSHL	SP		0361
	6B		01 FB 00086	CALLS	#1, DCLDIET\$SKIP_WHITESPACE		
			5E DD 00089	PUSHL	SP		0362
0000V	CF		01 FB 0008B	CALLS	#1, DCLDIET\$SCRUNCH_LINE		
	09		53 E8 00090	BLBS	CONTINUATION_LINE, 7\$		0363
			6E B6 00093	INCW	WORK_DSC		0364
		04	AE D7 00095	DECL	WORK_DSC+4		0365
04	BE		24 90 00098	MOVB	#36, @WORK_DSC+4		0366
	01		6E B1 0009C	CMPW	WORK_DSC, #1		0373
			06 1A 0009F	BGTRU	8\$		
	03		53 E8 000A1	BLBS	CONTINUATION_LINE, 8\$		0374
	36		56 E8 000A4	BLBS	DCL_COMMAND, 11\$		0375
	0B		56 E9 000A7	BLBC	DCL_COMMAND, 9\$		0376
05FC	C9		6E B0 000AA	MOVW	WORK_DSC, OUTPUT_RAB+34		0377
0602	C9	04	AE D0 000AF	MOVL	WORK_DSC+4, OUTPUT_RAB+40		0378
		05DA	C9 9F 000B5	PUSHAB	OUTPUT_RAB		0380
00000000G	00		01 FB 000B9	CALLS	#1, SYSSPUT		
	55		50 D0 000C0	MOVL	R0, STATUS		
	14		55 E8 000C3	BLBS	STATUS, 10\$		0381
		05E6	C9 DD 000C6	PUSHL	OUTPUT_RAB+12		
			55 DD 000CA	PUSHL	STATUS		
		22	A9 9F 000CC	PUSHAB	OUTPUT_ESA		
			01 DD 000CF	PUSHL	#1		
		000310D2	8F DD 000D1	PUSHL	#200914		
6A			05 FB 000D7	CALLS	#5, LIB\$SIGNAL		
13			56 E9 000DA	BLBC	DCL_COMMAND, 12\$		0389
50			6E 3C 000DD	MOVZWL	WORK_DSC, R0		0391
50		04	AE C0 000E0	ADDL2	WORK_DSC+4, R0		
			51 D4 000E4	CLRL	R1		
2D		FF	A0 91 000E6	CMPB	-1(R0), #45		
			06 12 000EA	BNEQ	13\$		
			51 D6 000EC	INCL	R1		
			02 11 000EE	BRB	13\$		
			51 D4 000F0	CLRL	R1		0389
53			51 90 000F2	MOVB	R1, CONTINUATION_LINE		
03			56 E8 000F5	BLBS	DCL_COMMAND, 14\$		0400
		00A4	31 000F8	BRW	21\$		
			50 D4 000FB	CLRL	R0		0401
05			6E B1 000FD	CMPW	WORK_DSC, #5		
			02 12 00100	BNEQ	15\$		
			50 D6 00102	INCL	R0		
			52 D4 00104	CLRL	R2		0402
24		04	BE 91 00106	CMPB	@WORK_DSC+4, #36		
			02 12 0010A	BNEQ	16\$		
			52 D6 0010C	INCL	R2		
51			50 D2 0010E	MCOML	R0, R1		
52			51 CA 00111	BICL2	R1, R2		

	50	04	AE	D0	00114	MOVL	WORK_DSC+4, R0		0403
	50	01	A0	9A	00118	MOVZBL	1(R0), R0		
	50	FFFFFF20	8F	CA	0011C	BICL2	#-224, R0		
00000044	8F		51	D4	00123	CLRL	R1		
			50	D1	00125	CMPL	R0, #68		
			02	12	0012C	BNEQ	17\$		
			51	D6	0012E	INCL	R1		
	57		52	D2	00130	MCOML	R2, R7		
	51		57	CA	00133	BICL2	R7, R1		
	50	04	AE	D0	00136	MOVL	WORK_DSC+4, R0		0404
	50	02	A0	9A	0013A	MOVZBL	2(R0), R0		
	50	FFFFFF20	8F	CA	0013E	BICL2	#-224, R0		
00000045	8F		52	D4	00145	CLRL	R2		
			50	D1	00147	CMPL	R0, #69		
			02	12	0014E	BNEQ	18\$		
			52	D6	00150	INCL	R2		
	58		51	D2	00152	MCOML	R1, R8		
	52		58	CA	00155	BICL2	R8, R2		
	50	04	AE	D0	00158	MOVL	WORK_DSC+4, R0		0405
	50	03	A0	9A	0015C	MOVZBL	3(R0), R0		
	50	FFFFFF20	8F	CA	00160	BICL2	#-224, R0		
00000043	8F		51	D4	00167	CLRL	R1		
			50	D1	00169	CMPL	R0, #67		
			02	12	00170	BNEQ	19\$		
			51	D6	00172	INCL	R1		
	57		52	D2	00174	MCOML	R2, R7		
	51		57	CA	00177	BICL2	R7, R1		
	50	04	AE	D0	0017A	MOVL	WORK_DSC+4, R0		0406
	50	04	A0	9A	0017E	MOVZBL	4(R0), R0		
	50	FFFFFF20	8F	CA	00182	BICL2	#-224, R0		
0000004B	8F		52	D4	00189	CLRL	R2		
			50	D1	0018B	CMPL	R0, #75		
			02	12	00192	BNEQ	20\$		
			52	D6	00194	INCL	R2		
54	58		51	D2	00196	MCOML	R1, R8		
	52		58	8B	00199	BICB3	R8, R2, WITHIN_DECK		0401
			5E	11	0019D	BRB	22\$		0407
	58		54	E9	0019F	BLBC	WITHIN_DECK, 22\$		0408
	24	04	BE	91	001A2	CMPL	@WORK_DSC+4, #36		
			55	12	001A6	BNEQ	22\$		
			6E	B7	001A8	DECW	WORK_DSC		0409
		04	AE	D6	001AA	INCL	WORK_DSC+4		0410
			5E	DD	001AD	PUSHL	SP		0411
	6B		01	FB	001AF	CALLS	#1, DCLDIET\$SKIP_WHITESPACE		
	03		6E	B1	001B2	CMPL	WORK_DSC, #3		0412
			46	12	001B5	BNEQ	22\$		
	50	04	BE	9A	001B7	MOVZBL	@WORK_DSC+4, R0		0413
00000045	50	FFFFFF20	8F	CA	001BB	BICL2	#-224, R0		
	8F		50	D1	001C2	CMPL	R0, #69		
			32	12	001C9	BNEQ	22\$		
	50	04	AE	D0	001CB	MOVL	WORK_DSC+4, R0		0414
	50	01	A0	9A	001CF	MOVZBL	1(R0), R0		
	50	FFFFFF20	8F	CA	001D3	BICL2	#-224, R0		
0000004F	8F		50	D1	001DA	CMPL	R0, #79		
			1A	12	001E1	BNEQ	22\$		
	50	04	AE	D0	001E3	MOVL	WORK_DSC+4, R0		0415
	50	02	A0	9A	001E7	MOVZBL	2(R0), R0		


```
; Routine Size: 513 bytes,    Routine Base: $CODES + 013D
```



```
: 428 0423 1 %sbttl 'DCLDIET$SCRUNCH_LINE - Remove Comments and Squeeze Whitespace'
: 429 0424 1 ++
: 430 0425 1 Functional Description:
: 431 0426 1 This procedure is called to scrunch a line in the command procedure.
: 432 0427 1 It removes any comments from the line. It also squeezes any
: 433 0428 1 whitespace down to a single blank. Trailing whitespace is eliminated.
: 434 0429 1
: 435 0430 1 Formal Parameters:
: 436 0431 1 line_dsc Descriptor of line. Line is scrunched by rearranging
: 437 0432 1 line buffer and updating descriptor.
: 438 0433 1
: 439 0434 1 Returned Value:
: 440 0435 1 none
: 441 0436 1
: 442 0437 1 Notes:
: 443 0438 1 none
: 444 0439 1 --
: 445 0440 1
: 446 0441 1
: 447 0442 1 GLOBAL ROUTINE dcldiet$scrunch_line(line_dsc: ref descriptor)
: 448 0443 2 : novalue = BEGIN
: 449 0444 2
: 450 0445 2 local
: 451 0446 2 scan_dsc: descriptor,
: 452 0447 2 char: byte;
: 453 0448 2
: 454 0449 2
: 455 0450 2 ! If the line is null, forget it.
: 456 0451 2
: 457 0452 2 if .line_dsc[len] eqlu 0 then
: 458 0453 2 return;
: 459 0454 2
: 460 0455 2 ! Begin by copying the line descriptor so we can scan the line. Then reset
: 461 0456 2 ! the length of the descriptor to zero so we can rebuild the line.
: 462 0457 2
: 463 0458 2 build_descriptor(scan_dsc,.line_dsc[len],.line_dsc[ptr]);
: 464 0459 2 line_dsc[len] = 0;
: 465 0460 2
: 466 0461 2 ! Now we will sit in a loop in order to find the first quotation mark
: 467 0462 2 ! or exclamation point in the line. As we go, we'll compress any whitespace
: 468 0463 2 ! down to a single blank.
: 469 0464 2
: 470 0465 2 while .scan_dsc[len] gtru 0 do (
: 471 0466 3 char = ch$rchar(.scan_dsc[ptr]);
: 472 0467 3 if .char eqlu '"' or .char eqlu '!' then exitloop;
: 473 0468 3 decrement(scan_dsc[len]);
: 474 0469 3 increment(scan_dsc[ptr]);
: 475 0470 4 if .char eqlu ' ' or .char eqlu HT then (
: 476 0471 4 ch$wchar(' ',.line_dsc[ptr]+.line_dsc[len]);
: 477 0472 4 dcldiet$skip_whitespace(scan_dsc);
: 478 0473 3 ) else
: 479 0474 3 ch$wchar(.char,.line_dsc[ptr]+.line_dsc[len]);
: 480 0475 3 increment(line_dsc[len]);
: 481 0476 2 );
: 482 0477 2
: 483 0478 2 ! If we found a quotation mark, we have some more work to do.
: 484 0479 2
```



```

: 485 0480 3 if .char eqlu "" then (
: 486 0481 3
: 487 0482 3     ! We have to be careful with the rest of the line, because the
: 488 0483 3     ! quotation rules in DCL are haphazard. Begin by moving down
: 489 0484 3     ! the rest of the line so it's adjacent to the part we compressed.
: 490 0485 3
: 491 0486 3     ch$move(.scan_dsc[.len],.scan_dsc[ptr],.line_dsc[ptr]+.line_dsc[.len]);
: 492 0487 3     line_dsc[.len]=.line_dsc[.len]+.scan_dsc[.len];
: 493 0488 3
: 494 0489 3     ! If there is an exclamation point at the end of the line, not
: 495 0490 3     ! followed by a quotation mark, then it's a comment.
: 496 0491 3
: 497 0492 3     scan_dsc[ptr] = .line_dsc[ptr] + .line_dsc[.len] - 1;
: 498 0493 3     while .scan_dsc[ptr] geqa .line_dsc[ptr] and
: 499 0494 3         ch$rchar(.scan_dsc[ptr]) nequ "" do (
: 500 0495 3         if ch$rchar(.scan_dsc[ptr]) eqlu "!" then (
: 501 0496 3             line_dsc[.len] = .scan_dsc[ptr] - .line_dsc[ptr];
: 502 0497 3             exitloop;
: 503 0498 3         );
: 504 0499 3         decrement(scan_dsc[ptr]);
: 505 0500 3     );
: 506 0501 3 );
: 507 0502 3
: 508 0503 3 ! If, after all that work, we ended up with trailing whitespace, then
: 509 0504 3 ! get rid of it.
: 510 0505 3
: 511 0506 3 scan_dsc[ptr] = .line_dsc[ptr] + .line_dsc[.len] - 1;
: 512 0507 3 while .scan_dsc[ptr] geqa .line_dsc[ptr] and
: 513 0508 3     (ch$rchar(.scan_dsc[ptr]) eqlu " " or ch$rchar(.scan_dsc[ptr]) eqlu HT) do (
: 514 0509 3     decrement(line_dsc[.len]);
: 515 0510 3     decrement(scan_dsc[ptr]);
: 516 0511 3 );
: 517 0512 3
: 518 0513 2 return;
: 519 0514 1 END;

```

			00FC 00000	.ENTRY	DCLDIET\$SCRUNCH_LINE, Save R2,R3,R4,R5,R6,-	0442
				R7		
5E		08	C2 00002	SUBL2	#8, SP	
56	04	AC	D0 00005	MOVL	LINE_DSC, R6	0452
		66	B5 00009	TSTW	(R6)	
		01	12 0000B	BNEQ	1\$	
			04 0000D	RET		
6E		66	3C 0000E 1\$:	MOVZWL	(R6), SCAN_DSC	0458
57	04	A6	9E 00011	MOVAB	4(R6), R7	
04	AE	67	D0 00015	MOVL	(R7), SCAN_DSC+4	
		66	B4 00019	CLRW	(R6)	0459
		6E	B5 0001B 2\$:	TSTW	SCAN_DSC	0465
		3C	13 0001D	BEQL	6\$	
52	04	BE	90 0001F	MOVB	@SCAN_DSC+4, CHAR	0466
22		52	91 00023	CMPB	CHAR, #34	0467
		33	13 00026	BEQL	6\$	
21		52	91 00028	CMPB	CHAR, #33	

			2E 13 0002B	BEQL	6\$		
			6E B7 0002D	DECW	SCAN_DSC		0468
		04	AE D6 0002F	INCL	SCAN_DSC+4		0469
	20		52 91 00032	CMPB	CHAR, #32		0470
			05 13 00035	BEQL	3\$		
	09		52 91 00037	CMPB	CHAR, #9		
			12 12 0003A	BNEQ	4\$		
	50		66 3C 0003C 3\$:	MOVZWL	(R6), R0		0471
	50		67 C0 0003F	ADDL2	(R7), R0		
	60		20 90 00042	MOVB	#32, (R0)		
			5E DD 00045	PUSHL	SP		0472
0000V	CF		01 FB 00047	CALLS	#1, DCLDIET\$SKIP_WHITESPACE		
			09 11 0004C	BRB	5\$		0474
	50		66 3C 0004E 4\$:	MOVZWL	(R6), R0		
	50		67 C0 00051	ADDL2	(R7), R0		
	60		52 90 00054	MOVB	CHAR, (R0)		
			66 B6 00057 5\$:	INCW	(R6)		0475
			C0 11 00059	BRB	2\$		0465
	22		52 91 0005B 6\$:	CMPB	CHAR, #34		0480
			37 12 0005E	BNEQ	9\$		
	50		66 3C 00060	MOVZWL	(R6), R0		0486
	50		67 C0 00063	ADDL2	(R7), R0		
60	04	BE	6E 28 00066	MOVCL3	SCAN_DSC, @SCAN_DSC+4, (R0)		
	66		6E A0 0006B	ADDW2	SCAN_DSC, (R6)		0487
	50		66 3C 0006E	MOVZWL	(R6), R0		0492
	50		67 C0 00071	ADDL2	(R7), R0		
	04	AE	FF A0 9E 00074	MOVAB	-1(R0), SCAN_DSC+4		
	67	04	AE D1 00079 7\$:	CMPL	SCAN_DSC+4, (R7)		0493
			18 1F 0007D	BLSSU	9\$		
	22	04	BE 91 0007F	CMPB	@SCAN_DSC+4, #34		0494
			12 13 00083	BEQL	9\$		
	21	04	BE 91 00085	CMPB	@SCAN_DSC+4, #33		0495
			07 12 00089	BNEQ	8\$		
66	04	AE	67 A3 0008B	SUBW3	(R7), SCAN_DSC+4, (R6)		0496
			05 11 00090	BRB	9\$		0495
		04	AE D7 00092 8\$:	DECL	SCAN_DSC+4		0499
			E2 11 00095	BRB	7\$		0493
	50		66 3C 00097 9\$:	MOVZWL	(R6), R0		0506
	50		67 C0 0009A	ADDL2	(R7), R0		
	04	AE	FF A0 9E 0009D	MOVAB	-1(R0), SCAN_DSC+4		
	67	04	AE D1 000A2 10\$:	CMPL	SCAN_DSC+4, (R7)		0507
			13 1F 000A6	BLSSU	12\$		
	20	04	BE 91 000A8	CMPB	@SCAN_DSC+4, #32		0508
			06 13 000AC	BEQL	11\$		
	09	04	BE 91 000AE	CMPB	@SCAN_DSC+4, #9		
			07 12 000B2	BNEQ	12\$		
			66 B7 000B4 11\$:	DECW	(R6)		0509
		04	AE D7 000B6	DECL	SCAN_DSC+4		0510
			E7 11 000B9	BRB	10\$		0507
			04 000BB 12\$:	RET			0514

; Routine Size: 188 bytes, Routine Base: \$CODE\$ + 033E


```
: 521 0515 1 %sbttl 'DCLDIET$SKIP_WHITESPACE - Skip Whitespace in a Line'
: 522 0516 1 ++
: 523 0517 1 Functional Description:
: 524 0518 1 This procedure is called to skip past the whitespace in a line.
: 525 0519 1 Whitespace is any sequence of blanks or tabs.
: 526 0520 1
: 527 0521 1 Formal Parameters:
: 528 0522 1 line_dsc Descriptor of line. Initial whitespace is skipped
: 529 0523 1 by updating descriptor.
: 530 0524 1
: 531 0525 1 Returned Value:
: 532 0526 1 none
: 533 0527 1
: 534 0528 1 Notes:
: 535 0529 1 none
: 536 0530 1 --
: 537 0531 1
: 538 0532 1
: 539 0533 1 GLOBAL ROUTINE dcldiet$skip_whitespace(line_dsc: ref descriptor)
: 540 0534 2 : novalue = BEGIN
: 541 0535 2
: 542 0536 2
: 543 0537 2 while .line_dsc[len] gtru 0 and
: 544 0538 3 (ch$rchar(.line_dsc[ptr]) eqlu ' ' or ch$rchar(.line_dsc[ptr]) eqlu HT) do (
: 545 0539 3 decrement(line_dsc[len]);
: 546 0540 3 increment(line_dsc[ptr]);
: 547 0541 2 );
: 548 0542 2
: 549 0543 2 return;
: 550 0544 1 END;
```

50	04	AC	D0	00002	1\$:	.ENTRY DCLDIET\$SKIP_WHITESPACE, Save nothing	: 0533
		60	B5	00006		MOVL LINE_DSC, R0	: 0537
		13	13	00008		TSTW (R0)	
20	04	B0	91	0000A		BEQL 3\$	
		06	13	0000E		CMPB @4(R0), #32	: 0538
09	04	B0	91	00010		BEQL 2\$	
		07	12	00014		CMPB @4(R0), #9	
		60	B7	00016	2\$:	BNEQ 3\$	
	04	A0	D6	00018		DECW (R0)	: 0539
		E9	11	0001B		INCL 4(R0)	: 0540
			04	0001D	3\$:	BRB 1\$	: 0537
						RET	: 0544

; Routine Size: 30 bytes, Routine Base: \$CODE\$ + 03FA

```
: 551 0545 1
: 552 0546 0 END ELUDOM
```

.EXTRN LIB\$SIGNAL

PSECT SUMMARY

Name	Bytes	Attributes
\$OWNS	3328	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)
\$PLITS	32	NOVEC, NOWRT, RD, NOEXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)
\$CODES	1048	NOVEC, NOWRT, RD, EXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
_\$255\$DUA28:[SYSLIB]STARLET.L32;1	9776	61	0	581	00:01.1

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LISS:DCLDIET/OBJ=OBJ\$:DCLDIET MSRC\$:DCLDIET/UPDATE=(ENHS:DCLDIET)

; Size: 1048 code + 3360 data bytes
; Run Time: 00:23.9
; Elapsed Time: 00:46.5
; Lines/CPU Min: 1370
; Lexemes/CPU-Min: 32610
; Memory Used: 183 pages
; Compilation Complete

0429 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

