

CCCCCCCCCCCC	DDDDDDDDDDDD	UUU	UUU
CCCCCCCCCCCC	DDDDDDDDDDDD	UUU	UUU
CCCCCCCCCCCC	DDDDDDDDDDDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCC	DDD	UUU	UUU
CCCCCCCCCCCC	DDDDDDDDDDDD	UUUUUUUUUUUUUUUU	
CCCCCCCCCCCC	DDDDDDDDDDDD	UUUUUUUUUUUUUUUU	
CCCCCCCCCCCC	DDDDDDDDDDDD	UUUUUUUUUUUUUUUU	

```

MM      MM      AAAAAA      IIIIII      NN      NN
MM      MM      AAAAAA      IIIIII      NN      NN
MMMM    MMMM    AA          AA      II      NN      NN
MMMM    MMMM    AA          AA      II      NN      NN
MM      MM      AA          AA      II      NNNN     NN
MM      MM      AA          AA      II      NNNN     NN
MM      MM      AA          AA      II      NN      NN      NN
MM      MM      AA          AA      II      NN      NN      NN
MM      MM      AAAAAAAAAA     II      NN      NNNN
MM      MM      AAAAAAAAAA     II      NN      NNNN
MM      MM      AA          AA      II      NN      NN
MM      MM      AA          AA      II      NN      NN
MM      MM      AA          AA      IIIIII     NN      NN
MM      MM      AA          AA      IIIIII     NN      NN

```

....
....
....
....

```

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

```

```
1 0001 0 MODULE main (IDENT='V04-000',
2 0002 0 MAIN=CDU$MAIN,
3 0003 0 ADDRESSING_MODE(EXTERNAL=GENERAL))
4 0004 1 = BEGIN
5 0005 1
6 0006 1 *****
7 0007 1 *
8 0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY *
9 0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS. *
10 0010 1 * ALL RIGHTS RESERVED. *
11 0011 1 *
12 0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *
13 0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE *
14 0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER *
15 0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *
16 0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY *
17 0017 1 * TRANSFERRED. *
18 0018 1 *
19 0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE *
20 0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT *
21 0021 1 * CORPORATION. *
22 0022 1 *
23 0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS *
24 0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL. *
25 0025 1 *
26 0026 1 *
27 0027 1 *****
28 0028 1
29 0029 1 **
30 0030 1 Facility: Command Definition Utility, Main Module
31 0031 1
32 0032 1 Abstract: This module contains the main routines for the Command
33 0033 1 Definition Utility, formerly known as the Command Language
34 0034 1 Editor. The CDU is responsible for maintaining CLI Tables,
35 0035 1 which are images or object files containing the internal
36 0036 1 representation of DCL or MCR commands. The primary
37 0037 1 component of the CDU is a compiler which reads Command
38 0038 1 Language Definition (CLD) files and compiles them into the
39 0039 1 internal format. Other features allow the deletion and
40 0040 1 extraction of information from DCL Tables, plus other
41 0041 1 goodies.
42 0042 1
43 0043 1 Special thanks goes to Tim Halvorsen, who wrote the
44 0044 1 original CDU. It has been rewritten to make it a bit more
45 0045 1 flexible and easy to maintain, particularly in light of all
46 0046 1 the enhancements in VMS V4.
47 0047 1
48 0048 1 Environment: Native, User mode. The following privileges are required:
49 0049 1
50 0050 1 CMEXEC For fooling with P1 space.
51 0051 1
52 0052 1 Author: Paul C. Anagnostopoulos
53 0053 1 Creation: 18 January 1983
54 0054 1
55 0055 1 Modifications:
56 0056 1 --
57 0057 1
```

MAIN
V04-000

15 Sep-1984 23:43:43
14-Sep-1984 11:58:24

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[CDU.SRC]MAIN.B32;1

Page 2
(1)

```
: 58      0058 1  
: 59      0059 1 library 'sys$library:lib';  
: 60      0060 1 require 'cdureq';
```

```

: 62      0474 1  !      T A B L E   O F   C O N T E N T S
: 63      0475 1  !      -----
: 64      0476 1
: 65      0477 1 forward routine
: 66      0478 1      cdu$main,
: 67      0479 1      cdu$delete_mode: novalue,
: 68      0480 1      cdu$object_mode: novalue,
: 69      0481 1      cdu$replace_mode: novalue,
: 70      0482 1      cdu$symbols_mode: novalue,
: 71      0483 1      cdu$report_rms_error: novalue;
: 72      0484 1
: 73      0485 1
: 74      0486 1  !      E X T E R N A L   R E F E R E N C E S
: 75      0487 1  !      -----
: 76      0488 1
: 77      0489 1 external routine
: 78      0490 1      cdu$cld,
: 79      0491 1      cdu$close_symbol_table_file,
: 80      0492 1      cdu$delete_verb_name,
: 81      0493 1      cdu$free_all_nodes,
: 82      0494 1      cdu$generate_table_blocks,
: 83      0495 1      cdu$open_next_cld,
: 84      0496 1      cdu$prepare_input_table,
: 85      0497 1      cdu$prepare_listing_file,
: 86      0498 1      cdu$prepare_new_table,
: 87      0499 1      cdu$prepare_object_file,
: 88      0500 1      cdu$report_listing_trailer,
: 89      0501 1      cdu$write_object_file,
: 90      0502 1      cdu$write_output_table,
: 91      0503 1      cdu$write_symbol_table_file,
: 92      0504 1      cli$get_value,
: 93      0505 1      cli$present,
: 94      0506 1      str$trim;
: 95      0507 1
: 96      0508 1 external
: 97      0509 1      cdu$gl_cld_errors: long;

```

```
: 99      0510 1 !      G L O B A L   D A T A
: 100     0511 1 !      -----
: 101     0512 1 !
: 102     0513 1 ! The following item specifies the facility string to be used in object files
: 103     0514 1 ! or any other files we create.
: 104     0515 1
: 105     0516 1 global bind
: 106     0517 1      cdu$facility_string = dtext('VAX/VMS Command Definition Utility (V4-001)'): descriptor;
```

```

: 108      0518 1 : **
: 109      0519 1 : Description: This is the main routine of the Command Definition Utility.
: 110      0520 1 : It is responsible for determining which operating mode the
: 111      0521 1 : user has requested and invoking a routine for that mode.
: 112      0522 1 :
: 113      0523 1 : Parameters: None.
: 114      0524 1 :
: 115      0525 1 : Returns: Most severe status encountered during execution.
: 116      0526 1 :
: 117      0527 1 : Notes:
: 118      0528 1 : --
: 119      0529 1 :
: 120      0530 1 GLOBAL ROUTINE cdu$main
: 121      0531 2 = BEGIN
: 122      0532 2
: 123      0533 2 own
: 124      0534 2 worst_status: long initial(msg(cdu$_success));
```

```

: 126      0535 2 ! The following routine is the global condition handler. Its purpose is to
: 127      0536 2 ! save the worst status that is signalled during the execution of the CDU.
: 128      0537 2 ! It is this status that is returned to DCL.
: 129      0538 2
: 130      0539 2 ROUTINE condition_handler(signal_vector: ref vector[,long])
: 131      0540 2 = BEGIN
: 132      0541 2
: 133      0542 2 bind
: 134      0543 2     status = signal_vector[1]: long;
: 135      0544 2 own
: 136      0545 2     severity_map: vector[8,byte] initial(byte(2,0,3,1,4,4,4,4));
: 137      0546 2
: 138      0547 2 if .severity_map[.status<0,3,0>] gtru .severity_map[.worst_status<0,3,0>] then
: 139      0548 2     worst_status = .status;
: 140      0549 2
: 141      0550 2 return false;
: 142      0551 2
: 143      0552 2 END;

```

```

                                .TITLE MAIN
                                .IDENT  \V04-000\
                                .PSECT  $SPLITS,NOWRT,NOEXE,2
64  6E  61  6D  6D  6F  43  20  53  4D  56  2F  58  41  56  00000 P.AAB: .ASCII  \VAX/VMS Command Definition Utility (V4-0\
69  74  55  20  6E  6F  69  74  69  6E  69  66  65  44  20  0000F
                                30  2D  34  56  28  20  79  74  69  6C  0001E
                                00  29  31  30  00028
                                010E002B 0002C P.AAA: .ASCII  \01)\<0>
                                00000000' 00030 .LONG  17694763
                                .ADDRESS P.AAB
                                .PSECT  $OWNS,NOEXE,2
                                00000000G 00000 WORST_STATUS:
                                04  04  04  04  01  03  00  02  00004 SEVERITY_MAP:
                                .LONG  CDUS_SUCCESS
                                .BYTE  2, 0, 3, 1, 4, 4, 4, 4
                                CDUSFACILITY_STRING==
                                P.AAA
                                .EXTRN  CDUSCLD, CDUSCLOSE SYMBOL_TABLE_FILE
                                .EXTRN  CDUSDELETE VERB NAME
                                .EXTRN  CDUSFREE ACL_NODES
                                .EXTRN  CDUSENERATE_TABLE_BLOCKS
                                .EXTRN  CDUSOPEN NEXT CLD
                                .EXTRN  CDUSPREPARE_INPUT TABLE
                                .EXTRN  CDUSPREPARE_LISTING FILE
                                .EXTRN  CDUSPREPARE_NEW TABLE
                                .EXTRN  CDUSPREPARE_OBJECT FILE
                                .EXTRN  CDUSREPORT LISTING TRAILER
                                .EXTRN  CDUSWRITE_OBJECT FILE
                                .EXTRN  CDUSWRITE_OUTPUT TABLE
                                .EXTRN  CDUSWRITE_SYMBOL_TABLE FILE
                                .EXTRN  CLISGET VALUE, CLISPRESENT
                                .EXTRN  STRSTRIM, CDUSGL_CLD ERRORS
                                .EXTRN  CDUS_SUCCESS

```

.PSECT \$CODE\$,NOWRT,2

000C 00000 CONDITION HANDLER:

51	52	04	53	0000'	CF	9E	00002	.WORD	Save R2,R3	:	0539
50	62		AC		04	C1	00007	MOVAB	WORST_STATUS, R3	:	0543
	63		03		00	EF	0000C	ADDL3	#4, SIGNAL_VECTOR, R2	:	0547
		04	03		00	EF	00011	EXTZV	#0, #3, (R2), R1	:	
			03		00	EF	00011	EXTZV	#0, #3, WORST_STATUS, R0	:	
		04	A340		04	A341	91 00016	CMPB	SEVERITY_MAP[R1], SEVERITY_MAP[R0]	:	
			63		03	1B	0001D	BLEQU	1\$:	
					62	D0	0001F	MOVL	(R2), WORST_STATUS	:	0548
					50	D4	00022	CLRL	R0	:	0550
					04	00024	1\$:	RET		:	0552

; Routine Size: 37 bytes, Routine Base: \$CODE\$ + 0000

```

: 145 0553 2 ! Main routine.
: 146 0554 2 ! Establish a global condition handler to save the worst status that is
: 147 0555 2 ! signalled.
: 148 0556 2
: 149 0557 enable
: 150 0558 condition_handler;
: 151 0559
: 152 0560 ! Determine which operating mode the user wants. This is specified by a
: 153 0561 ! major qualifier on the SET COMMAND command, of which /REPLACE is the default.
: 154 0562
: 155 0563 if cli$present(dtext('DELETE')) then
: 156 0564 cdu$delete_mode()
: 157 0565 else if cli$present(dtext('OBJECT')) then
: 158 0566 cdu$object_mode()
: 159 0567 else if cli$present(dtext('SYMBOLS')) then
: 160 0568 cdu$symbols_mode()
: 161 0569 else
: 162 0570 cdu$replace_mode();
: 163 0571
: 164 0572 ! Return the worst status that was signalled, with the inhibit flag set.
: 165 0573
: 166 0574 return .worst_status + sts$m_inhib_msg;
: 167 0575
: 168 0576 1 END;

```

```

.PSECT $SPLITS,NOWRT,NOEXE,2
00 00 45 54 45 4C 45 44 00034 P.AAD: .ASCII \DELETE\<0><0>
010E0006 0003C P.AAC: .LONG 17694726
00000000' 00040 .ADDRESS P.AAD
00 00 54 43 45 4A 42 4F 00044 P.AAF: .ASCII \OBJECT\<0><0>
010E0006 0004C P.AAE: .LONG 17694726
00000000' 00050 .ADDRESS P.AAF
00 53 4C 4F 42 4D 59 53 00054 P.AAH: .ASCII \SYMBOLS\<0>
010E0007 0005C P.AAG: .LONG 17694727
00000000' 00060 .ADDRESS P.AAH

```

```

.PSECT $CODE$,NOWRT,2
0004 00000 .ENTRY CDU$MAIN, Save R2 : 0530
52 00000000G 00 9E 00002 MOVAB CLISPRESNT, R2 : 0531
6D 0044 CF DE 00009 MOVAL 5$, (FP) : 0563
0000' CF 9F 0000E PUSHAB P.AAC
62 01 FB 00012 CALLS #1, CLISPRESNT
07 50 E9 00015 BLBC R0, 1$
0000V CF 00 FB 00018 CALLS #0, CDU$DELETE_MODE : 0564
27 11 0001D BRB 4$
0000' CF 9F 0001F 1$: PUSHAB P.AAE : 0565
62 01 FB 00023 CALLS #1, CLISPRESNT
07 50 E9 00026 BLBC R0, 2$
0000V CF 00 FB 00029 CALLS #0, CDU$OBJECT_MODE : 0566
16 11 0002E BRB 4$
0000' CF 9F 00030 2$: PUSHAB P.AAG : 0567

```

	62		01	FB	00034		CALLS	#1, CLIPRESENT		
	07		50	E9	00037		BLBC	R0, 3\$		
	0000V	CF	00	FB	0003A		CALLS	#0, CDUS\$SYMBOLS_MODE		0568
			05	11	0003F		BRB	4\$		
50	0000V	CF	00	FB	00041	3\$:	CALLS	#0, CDUS\$REPLACE_MODE		0570
	0000'	CF	8F	C1	00046	4\$:	ADDL3	#268435456, WORST_STATUS, R0		0574
				04	00050		RET			0576
				0000	00051	5\$:	.WORD	Save nothing		0531
			7E	D4	00053		CLRL	-(SP)		
			5E	DD	00055		PUSHL	SP		
	7E		AC	7D	00057		MOVQ	4(AP), -(SP)		
	CF	04	03	FB	0005B		CALLS	#3, CONDITION_HANDLER		
	FF7B			04	00060		RET			

; Routine Size: 97 bytes, Routine Base: \$CODES + 0025

```

170 0577 1 |**
171 0578 1 | Description: This routine handles /DELETE mode, in which the user wants
172 0579 1 | to remove one or more verb names from the CLI table. We
173 0580 1 | retrieve the list of verb names and delete them from the
174 0581 1 | table, reporting any errors.
175 0582 1 |
176 0583 1 | Parameters: None.
177 0584 1 |
178 0585 1 | Returns: Nothing.
179 0586 1 |
180 0587 1 | Notes:
181 0588 1 | --
182 0589 1 |
183 0590 1 GLOBAL ROUTINE cdu$delete_mode : novalue
184 0591 2 = BEGIN
185 0592 2
186 0593 2 local
187 0594 2     status: long;
188 0595 2
189 0596 2
190 0597 2 ! Call a routine to prepare the input CLI table for modification.
191 0598 2
192 0599 2 cdu$prepare_input_table();
193 0600 2
194 0601 2 ! Loop through the list of verb names to be deleted.
195 0602 2
196 0603 2 loop (
197 0604 2
198 0605 2     ! We need a buffer with descriptor to get a verb name.
199 0606 2
200 P 0607 2     with_dbuffer(verb_name,32,
201 P 0608 2
202 P 0609 2         ! Get the next verb name in the list. Quit if there aren't
203 P 0610 2         ! any more.
204 P 0611 2
205 P 0612 2         status = cli$get_value(dtext('DELETE'),verb_name);
206 P 0613 2         if not .status then exitloop;
207 P 0614 2         str$trim(verb_name,verb_name,verb_name);
208 P 0615 2
209 P 0616 2         ! Call a routine to delete the verb name from the table.
210 P 0617 2
211 P 0618 2         status = cdu$delete_verb_name(verb_name);
212 P 0619 2         check(.status, msg(cdu$_nosuchverb),1,verb_name);
213 0620 2     );
214 0621 2 );
215 0622 2
216 0623 2 ! Write out the modified CLI table.
217 0624 2
218 0625 2 cdu$write_output_table();
219 0626 2
220 0627 2 return;
221 0628 2
222 0629 1 END;

```

```

00 00 45 54 45 4C 45 44 00064 P.AAJ: .ASCII \DELETE\<0><0>
                                010E0006, 0006C P.AAI: .LONG 17694726
                                00000000, 00070 .ADDRESS P.AAJ
                                .EXTRN CDU$_NOSUCHVERB
                                .PSECT $CODE$,NOWRT,2
                                0004 00000 .ENTRY CDU$DELETE_MODE, Save R2
00000000G 5E 28 C2 00002 00000 SUBL2 #40, SP
00000000G 00 00 FB 00005 1$: 0599
04 AE 08 AE 9E 0000F MOVAB VERB_NAME+8, VERB_NAME+4
                                5E DD 00014 PUSHL SP
                                0000' LF 9F 00016 PUSHAB P.AAI
00000000G 00 02 FB 0001A CALLS #2, CLISGET_VALUE
00000000G 52 50 D0 00021 MOVL R0, STATUS
00000000G 31 52 E9 00024 BLBC STATUS, 2$
                                5E DD 00027 PUSHL SP
                                04 AE 9F 00029 PUSHAB VERB_NAME
                                08 AE 9F 0002C PUSHAB VERB_NAME
00000000G 00 03 FB 0002F CALLS #3, STR$TRIM
                                5E DD 00036 PUSHL SP
00000000G 00 01 FB 00038 CALLS #1, CDU$DELETE_VERB_NAME
00000000G 52 50 D0 0003F MOVL R0, STATUS
00000000G C7 52 E8 00042 BLBS STATUS, 1$
                                5E DD 00045 PUSHL SP
                                01 DD 00047 PUSHL #1
                                00000000G 8F DD 00049 PUSHL #CDU$_NOSUCHVERB
00000000G 00 03 FB 0004F CALLS #3, LIB$SIGNAL
00000000G 00 B4 11 00056 BRB 1$
                                00000000G 00 FB 00058 2$: 0599
                                04 0005F CALLS #0, CDU$WRITE_OUTPUT_TABLE
                                RET 0625
                                0629

```

; Routine Size: 96 bytes, Routine Base: \$CODE\$ + 0086

```

224 0630 1 |++
225 0631 1 | Description: This routine handles /OBJECT mode, in which the user wants
226 0632 1 | to compile an object file representing one CLD file. The
227 0633 1 | CLD file is compiled and the resulting table blocks are
228 0634 1 | written into an object file.
229 0635 1 |
230 0636 1 | Parameters: None.
231 0637 1 |
232 0638 1 | Returns: Nothing.
233 0639 1 |
234 0640 1 | Notes:
235 0641 1 | --
236 0642 1 |
237 0643 1 GLOBAL ROUTINE cdu$object_mode : novalue
238 0644 2 = BEGIN
239 0645 2
240 0646 2 local
241 0647 2     cld_fab: pointer,
242 0648 2     first_cld: boolean initial(true);
243 0649 2
244 0650 2
245 0651 2 ! Call a routine to set up a new, empty CLI table. Commands defined in the
246 0652 2 ! CLD file will be added to this table.
247 0653 2
248 0654 2 cdu$prepare_new_table();
249 0655 2
250 0656 2 ! Open the CLD file. If there isn't one, forget it.
251 0657 2
252 0658 2 cld_fab = cdu$open_next_cld();
253 0659 2 if .cld_fab eqla 0 then
254 0660 2     return;
255 0661 2
256 0662 2 ! Prepare the object file to receive the table blocks.
257 0663 2
258 0664 2 cdu$prepare_object_file(.cld_fab);
259 0665 2
260 0666 2 ! Prepare the listing file, if any, to receive the listing.
261 0667 2
262 0668 2 cdu$prepare_listing_file(.cld_fab);
263 0669 2
264 0670 2 ! Parse the CLD file into an intermediate representation.
265 0671 2
266 0672 2 cdu$cld();
267 0673 2
268 0674 2 ! If no syntax errors were discovered, then generate all of the CLI
269 0675 2 ! table blocks from the intermediate representation.
270 0676 2
271 0677 2 if .cdu$gl_cld_errors eglu 0 then
272 0678 2     cdu$generate_table_blocks();
273 0679 2
274 0680 2 ! If no errors of any kind were discovered, then write the object file.
275 0681 2
276 0682 2 if .cdu$gl_cld_errors eglu 0 then
277 0683 2     cdu$write_object_file();
278 0684 2
279 0685 2 ! Finish up the listing file.
280 0686 2

```

```

: 281      0687 2 cdu$report_listing_trailer();
: 282      0688 2
: 283      0689 2 return;
: 284      0690 2
: 285      0691 1 END;

```

```

                                000C 00000
                                53 00000000G 00 9E 00002
                                50          01 90 00009
00000000G 00          00 FB 0000C
00000000G 00          00 FB 00013
                                52          50 D0 0001A
                                36 13 0001D
                                52 DD 0001F
00000000G 00          01 FB 00021
                                52 DD 00028
00000000G 00          01 FB 0002A
00000000G 00          00 FB 00031
                                63 D5 00038
                                07 12 0003A
00000000G 00          00 FB 0003C
                                63 D5 00043 1$:
                                07 12 00045
00000000G 00          00 FB 00047
00000000G 00          00 FB 0004E 2$:
                                04 00055 3$:

.ENTRY CDU$OBJECT_MODE, Save R2,R3
MOVAB  CDU$GL_CLD_ERRORS, R3
MOVB   #1, FIRST_CLD
CALLS  #0, CDU$PREPARE_NEW_TABLE
CALLS  #0, CDU$OPEN_NEXT_CLD
MOVL   R0, CLD_FAB
BEQL   3$
PUSHL  CLD_FAB
CALLS  #1, -CDU$PREPARE_OBJECT_FILE
PUSHL  CLD_FAB
CALLS  #1, -CDU$PREPARE_LISTING_FILE
CALLS  #0, CDU$CLD
TSTL   CDU$GL_CLD_ERRORS
BNEQ   1$
CALLS  #0, CDU$GENERATE_TABLE_BLOCKS
TSTL   CDU$GL_CLD_ERRORS
BNEQ   2$
CALLS  #0, CDU$WRITE_OBJECT_FILE
CALLS  #0, CDU$REPORT_LISTING_TRAILER
RET

```

; Routine Size: 86 bytes, Routine Base: \$CODE\$ + 00E6

```

287 0692 1 |++
288 0693 1 | Description: This routine handles /REPLACE mode, which is the fundamental
289 0694 1 | mode by which a user adds or replaces command definitions.
290 0695 1 | We compile a set of CLD files and add/replace the
291 0696 1 | definitions to an existing CLI table specified by the user.
292 0697 1 | When compilation is complete, we create a new CLI table
293 0698 1 | with all the resulting definitions.
294 0699 1 |
295 0700 1 | Parameters: None.
296 0701 1 |
297 0702 1 | Returns: Nothing.
298 0703 1 |
299 0704 1 | Notes:
300 0705 1 | --
301 0706 1 |
302 0707 1 GLOBAL ROUTINE cdu$replace_mode : novalue
303 0708 2 = BEGIN
304 0709 2
305 0710 2 local
306 0711 2 cld_fab: pointer,
307 0712 2 errors: boolean initial(false);
308 0713 2
309 0714 2
310 0715 2 ! Call a routine to prepare the input CLI table for modification.
311 0716 2
312 0717 2 cdu$prepare_input_table();
313 0718 2
314 0719 2 ! Sit in a loop to compile each CLD file. Open each file in turn, quitting
315 0720 2 ! when we run out of files.
316 0721 2
317 0722 2 while (cld_fab = cdu$open_next_cld()) neqa 0 do (
318 0723 2
319 0724 2 ! Prepare the listing file, if any, to receive the listing.
320 0725 2
321 0726 2 cdu$prepare_listing_file(.cld_fab);
322 0727 2
323 0728 2 ! Parse the CLD file into its intermediate representation.
324 0729 2
325 0730 2 cdu$cld();
326 0731 2
327 0732 2 ! If no syntax errors were discovered, then generate all of the CLI
328 0733 2 ! table blocks from the intermediate representation.
329 0734 2
330 0735 2 if .cdu$gl_cld_errors eq 0 then
331 0736 2 cdu$generate_table_blocks();
332 0737 2
333 0738 2 ! Remember if any errors occurred, so we won't write the new table.
334 0739 2
335 0740 2 if .cdu$gl_cld_errors nequ 0 then
336 0741 2 errors = true;
337 0742 2
338 0743 2 ! Clear away the intermediate representation to prepare for the
339 0744 2 ! next CLD file.
340 0745 2
341 0746 2 cdu$free_all_nodes();
342 0747 2
343 0748 2 ! Finish up the listing file.

```

```

: 344      0749      3
: 345      0750      3      cdu$report_listing_trailer();
: 346      0751      3      );
: 347      0752      3
: 348      0753      3      ! If no errors were discovered, then write out the new CLI table.
: 349      0754      3
: 350      0755      3      if not .errors then
: 351      0756      3          cdu$write_output_table();
: 352      0757      3
: 353      0758      3      return;
: 354      0759      3
: 355      0760      3      1 END;

```

```

: 0707      .ENTRY      CDUSREPLACE MODE, Save R2,R3,R4
: 0708      MOVAB      CDUSGL_CLD_ERRORS, R4
: 0717      CLRB      ERRORS
: 0722      CALLS      #0, CDUSPREPARE_INPUT_TABLE
: 0726      CALLS      #0, CDUSOPEN_NEXT_CLD
: 0730      MOVL      R0, CLD_FAB
: 0735      BEQL      4$
: 0736      PUSHL     CLD_FAB
: 0741      CALLS      #1, CDUSPREPARE_LISTING_FILE
: 0746      CALLS      #0, CDUSCLD
: 0750      TSTL      CDUSGL_CLD_ERRORS
: 0755      BNEQ      2$
: 0756      CALLS      #0, CDUSGENERATE_TABLE_BLOCKS
: 0760      TSTL      CDUSGL_CLD_ERRORS
:          BEQL      3$
:          MOVB      #1, ERRORS
:          CALLS      #0, CDUSFREE_ALL_NODES
:          CALLS      #0, CDUSREPORT_LISTING_TRAILER
:          BRB      1$
:          BLBS      ERRORS, 5$
:          CALLS      #0, CDUSWRITE_OUTPUT_TABLE
:          RET

```

; Routine Size: 91 bytes, Routine Base: \$CODE\$ + 013C

```

357 0761 1 |++
358 0762 1 | Description: This routine handles /SYMBOLS mode, in which the user wants to
359 0763 1 | generate a symbol table file from a set of CLD files. The
360 0764 1 | symbol table file is needed when commands make use of the
361 0765 1 | old CLI interface. The symbols define the qualifier and
362 0766 1 | keyword numbers for use with the old CLI callbacks.
363 0767 1 |
364 0768 1 | In this mode, no CLI table blocks are generated.
365 0769 1 |
366 0770 1 | Parameters: None.
367 0771 1 |
368 0772 1 | Returns: Nothing.
369 0773 1 |
370 0774 1 | Notes:
371 0775 1 | --
372 0776 1 |
373 0777 1 GLOBAL ROUTINE cdu$symbols_mode : novalue
374 0778 2 = BEGIN
375 0779 2
376 0780 2 local
377 0781 2     symbols_written: boolean initial(false);
378 0782 2
379 0783 2
380 0784 2 ! Sit in a loop to compile each CLD file. Open each file in turn, quitting
381 0785 2 ! when we run out of files.
382 0786 2
383 0787 2 while cdu$open_next_cld() neqa 0 do (
384 0788 3
385 0789 3     ! Parse the CLD file into an intermediate representation.
386 0790 3
387 0791 3     cdu$cld();
388 0792 3
389 0793 3     ! If no syntax errors were discovered, then add the symbols from
390 0794 3     ! this CLD to the symbol table file.
391 0795 3
392 0796 4     if .cdu$gl_cld_errors eqlu 0 then (
393 0797 4         cdu$write_symbol_table_file();
394 0798 4         symbols_written = true;
395 0799 4     );
396 0800 3
397 0801 3     ! Clear away the intermediate representation to prepare for the
398 0802 3     ! next CLD file.
399 0803 3
400 0804 3     cdu$free_all_nodes();
401 0805 2 );
402 0806 2
403 0807 2 ! Close out the symbol table file if we ever wrote any.
404 0808 2
405 0809 2 if .symbols_written then
406 0810 2     cdu$close_symbol_table_file();
407 0811 2
408 0812 2 return;
409 0813 2
410 0814 1 END;

```

		00G4 00000	.ENTRY	CDU\$SYMBOLS_MODE, Save R2	: 0777
		52 94 00002	CLRB	SYMBOLS WRITTEN	: 0778
00000000G	00	00 FB 00004 1\$:	CALLS	#0, CDU\$OPEN_NEXT_CLD	: 0787
		50 D5 0000B	TSTL	R0	:
		22 13 0000D	BEQL	3\$:
00000000G	00	00 FB 0000F	CALLS	#0, CDU\$CLD	: 0791
	00000000G	00 D5 00016	TSTL	CDU\$GL_CLD_ERRORS	: 0796
		0A 12 0001C	BNEQ	2\$:
00000000G	00	00 FB 0001E	CALLS	#0, CDU\$WRITE_SYMBOL_TABLE_FILE	: 0797
		01 90 00025	MOVB	#1, SYMBOLS WRITTEN	: 0798
00000000G	00	00 FB 00028 2\$:	CALLS	#0, CDU\$FREE_ALL_NODES	: 0804
		D3 11 0002F	BRB	1\$: 0787
	07	52 E9 00031 3\$:	BLBC	SYMBOLS WRITTEN, 4\$: 0809
00000000G	00	00 FB 00034	CALLS	#0, CDU\$CLOSE_SYMBOL_TABLE_FILE	: 0810
		04 0003B 4\$:	RET		: 0814

; Routine Size: 60 bytes, Routine Base: \$CODE\$ + 0197

```

412 0815 1 |++
413 0816 1 | Description: This routine is called to report an error from an RMS
414 0817 1 | operation.
415 0818 1 |
416 0819 1 | Parameters: message By value, a message status code used for the
417 0820 1 | first line of the message. It is assumed
418 0821 1 | to take a single !AS $FA0 argument, the file
419 0822 1 | spec.
420 0823 1 | rms_block By reference, a FAB or RAB which contains
421 0824 1 | the error status code.
422 0825 1 |
423 0826 1 | Returns: Nothing.
424 0827 1 |
425 0828 1 | Notes: This routine assumes that all FABs have associated NAM
426 0829 1 | blocks.
427 0830 1 | --
428 0831 1 |
429 0832 1 | GLOBAL ROUTINE cdu$report_rms_error(message: long,
430 0833 1 | rms_block: pointer) : novalue
431 0834 2 | = BEGIN
432 0835 2 |
433 0836 2 | local
434 0837 2 |     fab: pointer,
435 0838 2 |     nam: pointer,
436 0839 2 |     file_spec: descriptor;
437 0840 2 |
438 0841 2 |
439 0842 2 | ! Pick up a pointer to the FAB and NAM blocks.
440 0843 2 |
441 0844 2 | fab = (if .rms_block[fab$b_bid] eglu fab$c_bid then .rms_block else .rms_block[rab$l_fab]);
442 0845 2 | nam = .fab[fab$l_nam];
443 0846 2 |
444 0847 2 | ! We need to find a file spec which can be included in the first message
445 0848 2 | ! line. Use the one which is most complete.
446 0849 2 |
447 0850 2 | if .nam[nam$b_rsl] nequ 0 then
448 0851 3 |     build_descriptor(file_spec, .nam[nam$b_rsl], .nam[nam$l_rsa])
449 0852 2 | else if .nam[nam$b_esl] nequ 0 then
450 0853 3 |     build_descriptor(file_spec, .nam[nam$b_esl], .nam[nam$l_esa])
451 0854 2 | else
452 0855 3 |     build_descriptor(file_spec, .fab[fab$b_fns], .fab[fab$l_fna]);
453 0856 2 | str$trim(file_spec, file_spec, file_spec);
454 0857 2 |
455 0858 2 | ! Signal the error stored in the RMS block.
456 0859 2 |
457 0860 2 | if .rms_block[fab$b_bid] eglu fab$c_bid then
458 0861 3 |     signal(.message, 1, file_spec, .rms_block[fab$l_sts], .rms_block[fab$l_stv])
459 0862 2 | else
460 0863 3 |     signal(.message, 1, file_spec, .rms_block[rab$l_sts], .rms_block[rab$l_stv]);
461 0864 2 |
462 0865 2 | return;
463 0866 2 |
464 0867 1 | END;

```

		000C	00000	.ENTRY	CDU\$REPORT_RMS_ERROR, Save R2,R3	: 0832
5E		08	C2 00002	SUBL2	#8, SP	
52	08	AC	D0 00005	MOVL	RMS_BLOCK, R2	: 0844
		53	D4 00009	CLRL	R3	
03		62	91 0000B	CMPB	(R2), #3	
		07	12 0000E	BNEQ	1\$	
		53	D6 00010	INCL	R3	
51		52	D0 00012	MOVL	R2, FAB	
		04	11 00015	BRB	2\$	
51	3C	A2	D0 00017	1\$: MOVL	60(R2), FAB	
50	28	A1	D0 0001B	2\$: MOVL	40(FAB), NAM	: 0845
	03	A0	95 0001F	TSTB	3(NAM)	: 0850
		0E	13 00022	BEQL	3\$	
6E	03	A0	9B 00024	MOVZBW	3(NAM), FILE_SPEC	: 0851
	02	AE	B4 00028	CLRW	FILE_SPEC+2	
04	AE	04	A0 D0 0002B	MOVL	4(NAM), FILE_SPEC+4	
		1F	11 00030	BRB	5\$: 0850
		0B	A0 95 00032	3\$: TSTB	11(NAM)	: 0852
		0E	13 00035	BEQL	4\$	
6E	0B	A0	9B 00037	MOVZBW	11(NAM), FILE_SPEC	: 0853
	02	AE	B4 0003B	CLRW	FILE_SPEC+2	
04	AE	0C	A0 D0 0003E	MOVL	12(NAM), FILE_SPEC+4	
		0C	11 00043	BRB	5\$: 0852
6E	34	A1	9B 00045	4\$: MOVZBW	52(FAB), FILE_SPEC	: 0855
	02	AE	B4 00049	CLRW	FILE_SPEC+2	
04	AE	2C	A1 D0 0004C	MOVL	44(FAB), FILE_SPEC+4	
		5E	DD 00051	5\$: PUSHL	SP	: 0856
	04	AE	9F 00053	PUSHAB	FILE_SPEC	
	08	AE	9F 00056	PUSHAB	FILE_SPEC	
00000000G	00	03	FB 00059	CALLS	#3, STR\$TRIM	
	7E	08	A2 7D 00060	MOVQ	8(R2), -(SP)	: 0863
		08	AE 9F 00064	PUSHAB	FILE_SPEC	
		01	DD 00067	PUSHL	#1	
00000000G	00	04	AC DD 00069	PUSHL	MESSAGE	
		05	FB 0006C	CALLS	#5, LIB\$SIGNAL	
		04	00073	RET		: 0867

: Routine Size: 116 bytes, Routine Base: \$CODE\$ + 01D3

: 465 0868 1 END
: 466 0869 0 ELUDOM

.EXTRN LIB\$SIGNAL

PSECT SUMMARY

Name	Bytes	Attributes
\$PLITS	116	NOVEC, NOWRT, RD, NOEXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)
\$OWNS	12	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)
\$CODE\$	583	NOVEC, NOWRT, RD, EXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
:_\$255\$DU428:[SYSLIB]LIB.L32;1	18619	19	0	1000	00:01.9

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:MAIN/OBJ=OBJ\$:MAIN MSRCS:MAIN/UPDATE=(ENHS:MAIN)

: Size: 583 code + 128 data bytes
: Run Time: 00:14.5
: Elapsed Time: 00:53.0
: Lines/CPU Min: 3608
: Lexemes/CPU-Min: 17373
: Memory Used: 116 pages
: Compilation Complete

