

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

```

NN      NN      000000      DDDDDDD      EEEEEEEEE      SSSSSSS
NN      NN      000000      DDDDDDD      EEEEEEEEE      SSSSSSS
NN      NN      00      00      DD      DD      EE      SS
NN      NN      00      00      DD      DD      EE      SS
NNNN    NN      00      00      DD      DD      EE      SS
NNNN    NN      00      00      DD      DD      EE      SS
NN      NN      00      00      DD      DD      EEEEEEE      SSSSSS
NN      NN      00      00      DD      DD      EEEEEEE      SSSSSS
NN      NNNN    00      00      DD      DD      EE      SS
NN      NNNN    00      00      DD      DD      EE      SS
NN      NN      00      00      DD      DD      EE      SS
NN      NN      00      00      DD      DD      EE      SS
NN      NN      000000      DDDDDDD      EEEEEEEEE      SSSSSSS
NN      NN      000000      DDDDDDD      EEEEEEEEE      SSSSSSS

```

```

LL      IIIIII      SSSSSSS
LL      IIIIII      SSSSSSS
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      SSSSSSS
LLLLLLLLLL      IIIIII      SSSSSSS

```

```
1 0001 0 MODULE nodes (IDENT='V04-000',
2 0002 0 ADDRESSING_MODE(EXTERNAL=GENERAL))
3 0003 1 = BEGIN
4 0004 1
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 Facility: Command Definition Utility, Node Routines
30 0030 1
31 0031 1 Abstract: This module provides routines to create and manipulate nodes,
32 0032 1 which are the data structure used to build the intermediate
33 0033 1 representation of a CLD file. These nodes are linked together
34 0034 1 as a directed graph, and when the parsing of the CLD is
35 0035 1 complete, represent the entire meaning of the CLD.
36 0036 1
37 0037 1 Environment: Standard CDU environment.
38 0038 1
39 0039 1 Author: Paul C. Anagnostopoulos
40 0040 1 Creation: 30 November 1982
41 0041 1
42 0042 1 Modifications:
43 0043 1 --
44 0044 1
45 0045 1
46 0046 1 library 'sys$library:lib';
47 0047 1 require 'cdureq';
```

:	49	0461	1	:	T A B L E	O F	C O N T E N T S
:	50	0462	1	:	-----	---	-----
:	51	0463	1	:			
:	52	0464	1	:	forward routine		
:	53	0465	1	:	cdu\$create_node,		
:	54	0466	1	:	cdu\$free_all_nodes: novalue,		
:	55	0467	1	:	cdu\$lookup_child,		
:	56	0468	1	:	cdu\$check_for_children;		
:	57	0469	1	:			
:	58	0470	1	:			
:	59	0471	1	:	E X T E R N A L	R E F E R E N C E S	
:	60	0472	1	:	-----	-----	
:	61	0473	1	:			
:	62	0474	1	:	external routine		
:	63	0475	1	:	lib\$get_vm;		
:	64	0476	1	:			
:	65	0477	1	:	external		
:	66	0478	1	:	cdu\$gl_line_number: long;		

```

: 68      0479 1  :      N O D E   S P A C E   D A T A
: 69      0480 1  :      -----
: 70      0481 1  :
: 71      0482 1  : ! The following items are needed to maintain the node space, which is a chunk
: 72      0483 1  : ! of memory that contains all of the nodes needed for the intermediate
: 73      0484 1  : ! representation of a CLD.
: 74      0485 1  :
: 75      0486 1  literal
: 76      0487 1      node_space_size = 256 * 512;      ! Size of node space is 128K.
: 77      0488 1  own
: 78      0489 1      node_space: pointer initial(0), ! Address of node space.
: 79      0490 1      next_node: pointer;      ! Address of next available byte.
```

```

81      0491 1  !++
82      0492 1  Description: This routine is called to create and initialize a new node.
83      0493 1  |
84      0494 1  |
85      0495 1  Parameters:  type           By value, a the type of the new node.
86      0496 1  text_length  Optional, by value, the length of the text
87      0497 1  string for the new node.
88      0498 1  text           Optional, by reference, the text string for
89      0499 1  the new node.
90      0500 1  |
91      0501 1  Returns:    Address of new node.
92      0502 1  |
93      0503 1  Notes:      Upon first call, this routine allocates the node space.
94      0504 1  |--
95      0505 1  |
96      0506 1  GLOBAL ROUTINE cdu$create_node(type: long,
97      0507 1  text_length: long,
98      0508 1  text: pointer)
99      0509 2  = BEGIN
100     0510 2  |
101     0511 2  builtin
102     0512 2  nullparameter;
103     0513 2  |
104     0514 2  local
105     0515 2  status: long,
106     0516 2  real_text_length: long,
107     0517 2  new_node: ref node;
108     0518 2  |
109     0519 2  |
110     0520 2  ! If the node space has not been allocated, then do it now.
111     0521 2  |
112     0522 2  if .node_space eql 0 then (
113     0523 2  status = lib$get_vm(%ref(node_space_size),node_space);
114     0524 2  check(.status, .status);
115     0525 2  next_node = .node_space;
116     0526 2  );
117     0527 2  |
118     0528 2  ! Set up a variable to tell us the actual length of the text string that is
119     0529 2  ! to be placed in the node.
120     0530 2  |
121     0531 2  real_text_length = (if nullparameter(2) then 0 else .text_length);
122     0532 2  |
123     0533 2  ! Allocate the node as an integral number of longwords. Make sure it fits
124     0534 2  ! within the allocated node space.
125     0535 2  |
126     0536 2  new_node = .next_node;
127     0537 2  next_node = .next_node + round_up(%fieldexpand(node_t_text,0) + .real_text_length, 4);
128     0538 2  if .next_node gtr .node_space + node_space_size then
129     0539 2  signal(msg(cdu$_intnodespace));
130     0540 2  |
131     0541 2  ! Initialize the node.
132     0542 2  |
133     0543 2  new_node[node_w_type] = .type;
134     0544 2  new_node[node_w_line] = .cdu$gl_line_number;
135     0545 2  new_node[node_l_sister] = new_node[node_l_child] = new_node[node_l_code] = 0;
136     0546 2  new_node[node_b_text_length] = .real_text_length;
137     0547 2  ch$move(.real_text_length,.text, new_node[node_t_text]);

```


NODES
V04-000

K 4
15-Sep-1984 23:44:46
14-Sep-1984 11:58:24

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

Page 6
(4)

11	A6	10	A6	52	90	00080	MOVB	REAL_TEXT_LENGTH, 16(NEW_NODE)	:	0546
		0C	BC	52	28	00084	MOV C3	REAL_TEXT_LENGTH, @TEXT, -17(NEW_NODE)	:	0547
			50	56	D0	0008A	MOVL	NEW_NODE, R0	:	0551
				04	0008D		RET		:	0553

; Routine Size: 142 bytes, Routine Base: \$CODE\$ + 0000

```

: 145      0554 1 !++
: 146      0555 1 | Description: This routine is called after a CLD file has been completely
: 147      0556 1 | processed. It frees up all the node space for the next file.
: 148      0557 1 |
: 149      0558 1 | Parameters: None.
: 150      0559 1 |
: 151      0560 1 | Returns: Nothing.
: 152      0561 1 |
: 153      0562 1 | Notes:
: 154      0563 1 | --
: 155      0564 1 |
: 156      0565 1 GLOBAL ROUTINE cdu$free_all_nodes      : novalue
: 157      0566 2 = BEGIN
: 158      0567 2
: 159      0568 2
: 160      0569 2 ! Simply reinitialize the next node pointer.
: 161      0570 2
: 162      0571 2 next_node = .node_space;
: 163      0572 2
: 164      0573 2 return;
: 165      0574 2
: 166      0575 1 END;

```

```

          0000' CF      0000' CF      0000 00000      .ENTRY CDU$FREE ALL NODES, Save nothing      : 0565
          0000' CF      0000' CF      04 00002      MOVL  NODE_SPACE, NEXT_NODE      : 0571
          0000' CF      0000' CF      04 00009      RET      : 0575

```

; Routine Size: 10 bytes, Routine Base: \$CODE\$ + 008E

```

168 0576 1 !++
169 0577 1 ! Description: This routine is called to search all of the children of a
170 0578 1 ! given parent node. It looks for a node of the given type.
171 0579 1 ! In addition, if desired, it will also compare a text string
172 0580 1 ! to the text string in the children of matching type.
173 0581 1
174 0582 1 ! Parameters: parent By reference, the parent node.
175 0583 1 ! type By value, the type of node that is desired.
176 0584 1 ! text_length Optional, by value, the length of the text
177 0585 1 ! string to match.
178 0586 1 ! text Optional, by reference, the text string to
179 0587 1 ! match.
180 0588 1
181 0589 1 ! Returns: Address of first matching child, or zero if none.
182 0590 1
183 0591 1 ! Notes:
184 0592 1 ! --
185 0593 1
186 0594 1 GLOBAL ROUTINE cdu$lookup_child(parent: ref node,
187 0595 1 ! type: long,
188 0596 1 ! text_length: long,
189 0597 1 ! text: pointer)
190 0598 2 = BEGIN
191 0599 2
192 0600 2 local
193 0601 2 child: ref node;
194 0602 2
195 0603 2 builtin
196 0604 2 nullparameter;
197 0605 2
198 0606 2
199 0607 2 ! We use one of two scanning loops, depending upon whether a text string
200 0608 2 ! match is desired.
201 0609 2
202 0610 2 if nullparameter(3) then
203 0611 2
204 0612 2 ! No text string match is desired, so just look for a child of the
205 0613 2 ! correct node type.
206 0614 2
207 P 0615 2 scan_children(parent,child,
208 PP 0616 2 if .child[node_w_type] eqlu .type then
209 P 0617 2 return .child;
210 0618 2 )
211 0619 2
212 0620 2 else
213 0621 2
214 0622 2 ! A text string match is desired, so we must look for a child of
215 0623 2 ! the correct node type which contains the specified string.
216 0624 2
217 P 0625 2 scan_children(parent,child,
218 PP 0626 2 if ch$eql(.child[node_b_text_length],child[node_t_text],
219 PP 0627 2 .text_length,.text,%x'00') and
220 PP 0628 2 .child[node_w_type] eqlu .type then
221 P 0629 2 return .child;
222 0630 2 );
223 0631 2
224 0632 2 ! We did not find a match, so return zero.

```

```

: 225      0633 2
: 226      0634 2 return 0;
: 227      0635 2
: 228      0636 1 END;

```

						50	04	AC	D0	00002		.ENTRY	CDU\$LOOKUP_CHILD, Save R2,R3,R4		0594
						03		6C	91	00006		MOVL	PARENT, R0		0618
								05	1F	00009		CMPB	(AP), #3		0610
							0C	AC	D5	0000B		BLSSU	1\$		
								14	12	0000E		TSTL	12(AP)		
						54	08	A0	D0	00010	1\$:	BNEQ	3\$		
								35	13	00014	2\$:	MOVL	8(R0), CHILD		0618
08	AC					10		00	ED	00016		BEQL	7\$		
								23	13	0001C		CMPZV	#0, #16, (CHILD), TYPE		
						54	04	A4	D0	0001E		BEQL	5\$		
								F0	11	00022		MOVL	4(CHILD), CHILD		
						54	08	A0	D0	00024	3\$:	BRB	2\$		
								21	13	00028	4\$:	MOVL	8(R0), CHILD		0630
						50	10	A4	9A	0002A		BEQL	7\$		
0C	AC			11		A4		50	2D	0002E		MOVZBL	16(CHILD), R0		
								BC		00035		CMPC5	R0, 17(CHILD), #0, TEXT_LENGTH, @TEXT		
								0C	12	00037		BNEQ	6\$		
						10		00	ED	00039		CMPZV	#0, #16, (CHILD), TYPE		
								04	12	0003F		BNEQ	6\$		
						50		54	D0	00041	5\$:	MOVL	CHILD, R0		
									04	00044		RET			
						54	04	A4	D0	00045	6\$:	MOVL	4(CHILD), CHILD		
								DD	11	00049		BRB	4\$		
								50	D4	0004B	7\$:	CLRL	R0		0634
								04	0004D		RET				0636

; Routine Size: 78 bytes, Routine Base: \$CODE\$ + 0098

```

: 230      0637 1  !++
: 231      0638 1  ! Description: This routine is called to search all of the children of a given
: 232      0639 1  ! parent node, looking for children of certain types. The caller
: 233      0640 1  ! is only interested in knowing whether at least one of the
: 234      0641 1  ! specified node types exist.
: 235      0642 1  !
: 236      0643 1  ! Parameters: parent          By reference, the parent node.
: 237      0644 1  ! types...          By value, one or more node types to search for.
: 238      0645 1  !
: 239      0646 1  ! Returns:          By value, a boolean which is true if one of the types is found.
: 240      0647 1  !
: 241      0648 1  ! Notes:
: 242      0649 1  ! --
: 243      0650 1  !
: 244      0651 1  GLOBAL ROUTINE cdu$check_for_children(parent: ref node,
: 245      0652 1  !                                     types: vector[,long])
: 246      0653 2  = BEGIN
: 247      0654 2  !
: 248      0655 2  local
: 249      0656 2  !     types_size: long,
: 250      0657 2  !     child: ref node,
: 251      0658 2  !     type_key: long;
: 252      0659 2  !
: 253      0660 2  builtin
: 254      0661 2  !     actualcount;
: 255      0662 2  !
: 256      0663 2  !
: 257      0664 2  ! Calculate the size of the types list for use in the loop below.
: 258      0665 2  !
: 259      0666 2  types_size = (actualcount() - 1) * 4;
: 260      0667 2  !
: 261      0668 2  ! Scan all the children of the parent node.
: 262      0669 2  !
: 263      P 0670 2  scan_children(parent,child,
: 264      P 0671 2  !
: 265      P 0672 2  !     ! Lookup the node type in the type list passed to us. If found,
: 266      P 0673 2  !     ! return true.
: 267      P 0674 2  !
: 268      P 0675 2  !     type key = .child[node_w_type];
: 269      P 0676 2  !     if ch$find_sub(.types_size,types, 4,type_key) neqa 0 then
: 270      P 0677 2  !     return true;
: 271      0678 2  ! );
: 272      0679 2  !
: 273      0680 2  ! We didn't find a child of the specified types, so return false.
: 274      0681 2  !
: 275      0682 2  return false;
: 276      0683 2  !
: 277      0684 1  END;

```

```

                    003C 00000      .ENTRY  CDUSCHECK_FOR_CHILDREN, Save R2,R3,R4,R5      : 0651
5E                  04 C2 00002      SUBL2   #4, SP
50                  6C 9A 00005      MOVZBL (AP), R0
                    50 D7 00008      DECL   R0
                                     : 0666
                                     :

```

```

55          50      02 78 0000A      ASHL #2, R0, TYPES_SIZE
50          50      04 AC D0 0000E      MOVL PARENT, R0
54          08      08 A0 D0 00012      MOVL 8(R0), CHILD
6E          1D 13 00016 1$:      BEQL 4$
6E          64 3C 00018      MOVZWL (CHILD), TYPE_KEY
08 AC      55      6E          04 39 0001B      MATCHC #4, TYPE_KEY, TYPES_SIZE, TYPES
6E          03 13 00021      BEQL 2$
53          04 D0 00023      MOVL #4, R3
53          04 C2 00026 2$:      SUBL2 #4, R3
50          04 13 00029      BEQL 3$
50          01 D0 0002B      MOVL #1, R0
54          04 A4 D0 0002E      RET
          04      E1 11 00033      MOVL 4(CHILD), CHILD
          50      D4 00035 4$:      BRB 1$
          04      04 00037      CLRL R0
          RET
: 0678
: 0682
: 0684

```

: Routine Size: 56 bytes, Routine Base: \$CODE\$ + 00E6

```

: 278          0685 1 END
: 279          0686 0 ELUDOM

```

.EXTRN LIB\$SIGNAL

PSECT SUMMARY

Name	Bytes	Attributes
\$OWNS	8	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)
\$CODE\$	286	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]LIB.L32;1	18619	4	0	1000	00:01.9

COMMAND QUALIFIERS

: BLISS/CHECK=(FIELD, INITIAL, OPTIMIZE)/LIS=LIS\$:NODES/OBJ=OBJ\$:NODES MSRC\$:NODES/UPDATE=(ENH\$:NODES)

```

: Size:          286 code + 8 data bytes
: Run Time:      00:10.2

```

NODES
V04-000

^{0 5}
15-Sep-1984 23:44:46

VAX-11 Bliss-32 V4.0-742

Page 12

: Elapsed Time: 00:34.5
: Lines/CPU Min: 4047
: Lexemes/CPU-Min: 18035
: Memory Used: 100 pages
: Compilation Complete

SYMBOLS LIS
LISTING LIS
MAIN LIS
NODES LIS
OBJECT LIS
PARSE1 LIS
PARSE2 LIS
PARSE3 LIS
ROUTINES LIS
TABLE LIS