

```
CCCCCCCCCCCCC DDDDDDDDDDD UUU UUU
CCCCCCCCCCCCC DDDDDDDDDDD UUU UUU
CCCCCCCCCCCCC DDDDDDDDDDD UUU UUU
CCC           DDD           DDD UUU UUU
CCC           DDD           DDD UUU UUU
CCC           DDD           DDD UUU UUU
CCC           DDD           DDD UUU UUU
CCC           DDD           DDD UUU UUU
CCC           DDD           DDD UUU UUU
CCC           DDD           DDD UUU UUU
CCC           DDD           DDD UUU UUU
CCC           DDD           DDD UUU UUU
CCC           DDD           DDD UUU UUU
CCC           DDD           DDD UUU UUU
CCC           DDD           DDD UUU UUU
CCC           DDD           DDD UUU UUU
CCC           DDD           DDD UUU UUU
CCCCCCCCCCCCC DDDDDDDDDDD UUUUUUUUUUUUUUUUU
CCCCCCCCCCCCC DDDDDDDDDDD UUUUUUUUUUUUUUUUU
CCCCCCCCCCCCC DDDDDDDDDDD UUUUUUUUUUUUUUUUU
```

```
CCCCCCCC DDDDDDDD UU UU RRRRRRRR EEEEEEEEE EEEEEEEEE QQQQQQ
CCCCCCCC DDDDDDDD UU UU RRRRRRRR EEEEEEEEE EEEEEEEEE QQQQQQ
CC CC DD DD UU UU RR RR EE EE QQ QQ
CC CC DD DD UU UU RR RR EE EE QQ QQ
CC CC DD DD UU UU RR RR EE EE QQ QQ
CC CC DD DD UU UU RRRRRRRR EEEEEEEEE EEEEEEEEE QQ QQ
CC CC DD DD UU UU RRRRRRRR EEEEEEEEE EEEEEEEEE QQ QQ
CC CC DD DD UU UU RR RR EE EE QQ QQ
CC CC DD DD UU UU RR RR EE EE QQ QQ
CC CC DD DD UU UU RR RR EE EE QQ QQ
CCCCCCCC DDDDDDDD UUUUUUUUU RRR RR EEEEEEEEE EEEEEEEEE QQQQ QQ
CCCCCCCC DDDDDDDD UUUUUUUUU RRR RR EEEEEEEEE EEEEEEEEE QQQQ QQ
```

```
RRRRRRR 333333 222222
RRRRRRR 333333 222222
RR RR 33 33 22 22
RR RR 33 33 22 22
RR RR 33 33 22 22
RRRRRRR 33 22
RRRRRRR 33 22
RR RR 33 22
RR RR 33 22
RR RR 33 22
RR RR 333333 2222222222
RR RR 333333 2222222222
```



! IDENT='V04-000'

```

*****
* COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
* DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
* ALL RIGHTS RESERVED.
*
* THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
* ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
* INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
* COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
* OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
* TRANSFERRED.
*
* THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
* AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
* CORPORATION.
*
* DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
* SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
*
*****

```

```

++
Facility:      Command Definition Utility, Require File
Abstract:      This require file contains definitions which pertain
                specifically to the Command Definition Utility.
Environment:   Standard CDU environment.
Author:        Paul C. Anagnostopoulos
Creation:      29 November 1982
Modifications:
    V04-002 BLS0281      Benn Schreiber      4-MAR-1984
                Rename generalreq to 9 chars.
    V04-001 PCG0001      Peter George        06-Dec-1983
                Add NEG operator.
--

```

require 'genralreq';

```

!
!   L E X I C A L   A N A L Y S I S
!   -----

```

```

! The following items define the token class names. Each token which is
! isolated as a result of lexical analysis fits into one of the following
! classes.

```

```

literal
tkn_k_invalid =          0,      ! Invalid characters.
tkn_k_ignored =         1,      ! Characters which are ignored.
tkn_k_whitespace =     2,      ! Whitespace.
tkn_k_eol =             3,      ! End of line.
tkn_k_eof =            4,      ! End of file.
tkn_k_comma =          5,      ! Comma for list separation.
tkn_k_equal =          6,      ! Equal sign.
tkn_k_open_paren =     7,      ! Open parenthesis for grouping.
tkn_k_close_paren =   8,      ! Close parenthesis for grouping.
tkn_k_dot =            9,      ! Dot for path names.
tkn_k_comment =       10,      ! Comment delimiter.
tkn_k_string =        11,      ! Quoted string.
tkn_k_h_string =      12,      ! Quoted string, or arbitrary stuff
                                ! ending at end of line or various
                                ! other delimiters.

tkn_k_symbol =        13,      ! Symbol.
tkn_k_open_angle =    14,      ! Open angle bracket for paths.
tkn_k_close_angle =  15,      ! Close angle bracket for paths.
tkn_k_max_class =     15;      ! Must be highest class number.

literal
tkn_k_max_length =     255;     ! Maximum length of a token. Must fit
                                ! in an ASCII string.

```

```

! The following macro is used to test whether the current token is of a
! given class, and optionally, whether it matches a particular string.

```

```

macro
token_is(token_class,match_string) =
    %if %null(match_string) %then
        (.cdu$gl_token_class equl token_class)
    %else
        (.cdu$gl_token_class equl token_class and
         ch$eq(.cdu$gq_token[len],.cdu$gq_token[ptr],
              %charcount(match_string),uplit byte(match_string),%x'00'))
    %fi %;

```

```

! The following macro is used to skip over a token which is optional in the
! syntax. It allows an optional hint to the token routine.

```

```

macro
skip_optional_token(token_class,hint) =
    %if %null(hint) %then
        (if token_is(token_class) then
         cdu$get_next_token();)
    %else
        (if token_is(token_class) then
         cdu$get_next_token(hint);)

```



%fi %;

! INTERMEDIATE REPRESENTATION

! The following structure defines a node in the intermediate representation  
! created during syntactic analysis. These nodes are linked together as a  
! directed graph. The same node is used to build the symbol table.

```
field
  node_fields = set
    node_w_type =      [ 0,0,16,0],    ! Type of node.
    node_w_line =     [ 2,0,16,0],    ! CLD file line number.
    node_l_sister =    [ 4,0,32,0],    ! Right sister node.
    node_l_child =     [ 8,0,32,0],    ! Parent node.
    node_l_code =      [12,0,32,0],    ! Generated table code.
    node_b_text_length = [16,0, 8,0],  ! Length of text and
    node_t_text =      [17,0, 0,0]    ! the text string.
  tes;
```

! The following is a list of all of the node types.

```
literal
  node_k_error =      0,
  node_k_root =       1,
  node_k_ident =      2,
  node_k_module =     3,
  node_k_define_verb = 4,
  node_k_define_syntax = 5,
  node_k_define_type = 6,
  node_k_cliflags =   7,
  node_k_cliroutine = 8,
  node_k_disallow =   9,
  node_k_image =     10,
  node_k_outputs =   11,
  node_k_outputs_item = 12,
  node_k_parameter = 13,
  node_k_noparameters = 14,
  node_k_prefix =    15,
  node_k_qualifier = 16,
  node_k_noqualifiers = 17,
  node_k_routine =   18,
  node_k_prompt =    19,
  node_k_batch =     20,
  node_k_negatable = 21,
  node_k_nonnegatable = 22,
  node_k_placement = 23,
  node_k_keyword =   24,
  node_k_default =   25,
  node_k_label =     26,
  node_k_syntax =    27,
  node_k_value =     28,
  node_k_impct =     29,
  node_k_list =      30,
  node_k_required =  31,
  node_k_type_builtin = 32,
  node_k_type_user = 33,
  node_k_abbrev =    34,
```



```

node_k_foreign = 35,
node_k_immed = 36,
node_k_mcrignore = 37,
node_k_mcroptdelim = 38,
node_k_mcrparse = 39,
node_k_nostat = 40,
node_k_or = 41,
node_k_nodisallows = 42,
node_k_and = 43,
node_k_not = 44,
node_k_any2 = 45,
node_k_path = 46,
node_k_resolution = 47,
node_k_path_definition = 48,
node_k_path_entity = 49,
node_k_concatenate = 50,
node_k_noconcatenate = 51,
node_k_synonym = 52,
node_k_neg = 53,
node_k_max_type = 53;

```

! The following macro makes it easier to declare a node.

```

macro
  node = block[,byte] field(node_fields) %;

```

! The following macro is used to link a child node onto a parent node.  
! The child is linked at the end of any existing child chain, which is  
! why the last\_child pointer is required.

```

macro
  link_parent_to_child(parent,child,last_child) =
    (if .parent[node_l_child] egl 0 then
      last_child = parent[node_l_child] = .child
    else
      last_child = last_child[node_l_sister] = .child;
    ) %;

```

! The following macro is used to traverse all of the children of a parent node.  
! The child pointer is set to point at each child in turn.

```

macro
  scan_children(parent,child) =
    (child = .parent[node_l_child];
    while .child nega 0 do ?
      %remaining
      child = .child[node_l_sister];
    );
  ) %;

```

```
!
!   TABLE BLOCK GENERATION
!   -----
```

```
! The following macro is used to allocate space for the largest possible
! table block of a certain type. The block will then be filled in with
! information from the intermediate representation.
```

```
macro
  allocate_largest_table_block(size,ptr) =
    (local
      status: long;
      status = lib$get_vm(%ref(size),ptr);
      check(.status, .status);
    ) %;
```

```
! This macro is used to set the final size of a table block, rounded up to
! a longword boundary. The size is also added to the overall table size
! stored in the primary vector block.
```

```
macro
  set_table_block_size(size,ptr) =
    (ptr[cmd_w_size] = round_up(size,4);
     cdu$gl_table[vec_l_table_size] = .cdu$gl_table[vec_l_table_size] + .ptr[cmd_w_size];
    ) %;
```

```
! This macro is used to extend the size of a table block. The size
! increase is also added to the overall table size stored in the primary
! vector block. Blocks can be shrunk without worrying about the overall
! table size.
```

```
macro
  extend_table_block_size(extension,ptr) =
    (ptr[cmd_w_size] = .ptr[cmd_w_size] + extension;
     cdu$gl_table[vec_l_table_size] = .cdu$gl_table[vec_l_table_size] + extension;
    ) %;
```



GENRAL REQ R32	EXTCAL LIS
CLISDEF R32	GENCODE4 LIS
CDUMSGS LIS	GENCODE1 LIS
CDU	GENCODE2 LIS
CDU MAP	GENCODE3 LIS
CDUREQ R32	GENCODE2 LIS
CDU PDEF LIS	