80-13

80-13

A Description of the Display Module for Interactive Presentation of Visual Information on a Plasma Terminal (ORION)

Albert Boulanger

Internal Report

No. 80-13

Intelligent Systems Group

Department of Computer Science

University of Illinois

Urbana Illinois

This research is supported in part by the US Department of Agriculture grant No. 901-15-44 and in part by the National Science Foundation grant No. MCS 79-06614.

```
000003    (*$L'DISPLAY MODULE ROUTINES'*)
000003    (*===============================================================
000003    X===============================================================X
000003    X===============================================================X
000003    X===============================================================X
000003    X                                                               X
000003    X                                                               X
000003    X                                                               X
000003    X                                                               X
000003    X DDDDDD     IIII    SSSSSS  PPPPPP   LL        AAAAA   YY   YY X
000003    X DDDDDD     IIII    SSSSSS  PPPPPP   LL        AAAAA   YY   YY X
000003    X DD   DD     II    SS       PP   PP  LL       AA   AA   YY YY  X
000003    X DD   DD     II    SS       PP   PP  LL       AA   AA   YY YY  X
000003    X DD   DD     II    SS       PP   PP  LL       AA   AA    YYY   X
000003    X DD   DD     II    SS       PP   PP  LL       AA   AA    YYY   X
000003    X DD   DD     II     SSSSS   PPPPPP   LL       AAAAAAA     YY   X
000003    X DD   DD     II     SSSSS   PPPPPP   LL       AAAAAAA     YY   X
000003    X DD   DD     II         SS  PP       LL       AA   AA     YY   X
000003    X DD   DD     II         SS  PP       LL       AA   AA     YY   X
000003    X DD   DD     II         SS  PP       LL       AA   AA     YY   X
000003    X DD   DD     II         SS  PP       LL       AA   AA     YY   X
000003    X DDDDDD     IIII   SSSSSS   PP       LLLLLLL  AA   AA     YY   X
000003    X DDDDDD     IIII   SSSSSS   PP       LLLLLLL  AA   AA     YY   X
000003    X                                                               X
000003    X                                                               X
000003    X                                                               X
000003    X                                                               X
000003    X          M    M    OOOOO   DDDDDD   UU   UU  LL       EEEEEEE X
000003    X          M    M    OOOOO   DDDDDD   UU   UU  LL       EEEEEEE X
000003    X          MM   MM  OO   OO  DD   DD  UU   UU  LL       EE      X
000003    X          MM   MM  OO   OO  DD   DD  UU   UU  LL       EE      X
000003    X          MMM MMM  OO   OO  DD   DD  UU   UU  LL       EE      X
000003    X          MMM MMM  OO   OO  DD   DD  UU   UU  LL       EE      X
000003    X          MM M MM  OO   OO  DD   DD  UU   UU  LL       EEEEE   X
000003    X          MM M MM  OO   OO  DD   DD  UU   UU  LL       EEEEE   X
000003    X          MM   MM  OO   OO  DD   DD  UU   UU  LL       EE      X
000003    X          MM   MM  OO   OO  DD   DD  UU   UU  LL       EE      X
000003    X          MM   MM  OO   OO  DD   DD  UU   UU  LL       EE      X
000003    X          MM   MM  OO   OO  DD   DD  UU   UU  LL       EE      X
000003    X          MM   MM   OOOOO   DDDDDD    UUUUU   LLLLLLL  EEEEEEE X
000003    X          MM   MM   OOOOO   DDDDDD    UUUUU   LLLLLLL  EEEEEEE X
000003    X                                                               X
000003    X                                                               X
000003    X                                                               X
000003    X                                                               X
000003    X===============================================================X
000003    X===============================================================X
000003    X===============================================================X
000003    ===============================================================*)
000003
000003
000003    (*
000003    ----------------- I N T R O D U C T I O N ----------------
000003                      -----------------------
000003
000003       THIS SET OF PASCAL FUNCTIONS AND PROCEDURES ARE THE NECESSARY
000003    COMPONENTS OF THE DISPLAY MODULE. THIS SET OF ROUTINES WAS
```

WRITTEN BY ALBERT BOULANGER DURING THE SUMMER AND FALL OF 1980. THESE ROUTINES ARE TARGETED FOR THE MAGNAVOX ORION-60 PLASMA DISPLAY TERMINAL. EVENTUALLY THESE ROUTINES SHOULD SUPPORT SEVERAL TYPES OF CRT DISPLAY THAT HAVE AT LEAST CURSOR ADDRESSABILITY. A GOOD EXAMPLE IS THE TEKTRONIX 4012. THESE ROUTINES WERE WRITTEN TO BE USED IN THE CO-OPERATIVE KNOWLEDGE BASE EFFORT.

---------------- D E S I G N ----------------

1

MOTIVATION FOR IMPLEMENTING THE DISPLAY MODULE.

THE DISPLAY MODULE WAS DESIGNED TO PROVIDE A HUMAN ENGINEERED INTERFACE TO A KNOWLEDGE BASE SYSTEM. IN THE DISPLAY MODULE PACKAGE, THE MAJOR MEANS FOR A USER TO INTERACT WITH THE COMPUTER IS VIA THE TOUCH PANEL. THE MAJORITY OF GRAPHICS IN A KNOWLEDGE BASE SYSTEM IS TEXTUALLY RELATED. (I.E. DISPLAY AN INFERENCE TREE, OR A TABLE, ETC.) THEREFORE, THIS SYSTEM DOES SUPPORT GRAPHICS, BUT IT IS TEXTUALLY RELATED. THERE ARE NO CLIPPING OR WINDOWING OR TRANSFORMATION FACILITIES IN THE SYSTEM. IT WAS FELT THAT THERE WAS A NEED TO DISPLAY SEVERAL QUESTIONS OR SEVERAL RELATED TOPICS SIMULTANEOUSLY ON THE SCREEN TO MAXIMIZE USER RESPONSE BANDWITH. THE DISPLAY SCREEN CONTEXT IN WHICH SEVERAL QUESTIONS OR SEVERAL DIFFERENT TOPICS CAN BE PRESENTED IS A SET OF BLOCKS. A BLOCK IS A VIRTUAL TERMINAL. IDEALLY, A VIRTUAL TERMINAL BEHAVES LIKE A REAL TERMINAL. BLOCKS CAN BE NESTED. THE DISPLAY SCREEN DEVICE WHICH ALLOWS A USER TO RESPOND VIA THE TOUCH PANEL IS THE TOUCH TARGET. (HERE ON ABBREVIATED AS TT.) A TT IS AN AREA OF THE SCREEN THAT HAS ONE OR MORE TOUCH PANEL INTERSECTIONS IN IT. FUNCTIONALLY, A TT TELLS THE USER WHERE THE SENSITIVE AREA FOR A CERTIAN RESPONSE IS. WHEN TOUCHED, THE TT CHANGES ITS VISUAL CHARACTERISTICS TO GIVE THE USER FEEDBACK.

IN ORDER TO COMPLETE THE CONCEPT OF A VIRTUAL TERMINAL, MOST OF THE FUNCTIONS (EXCEPT BLOCK MAINTENANCE AND TT INPUT) ARE ACCESSED THROUGH A BLOCK DISPLAY PROCEDURE. THIS PROCEDURE IS PARAMETERISED IN PART BY A TEXT FILE TO DISPLAY AND A BLOCK IDENTIFIER. TTS ARE PLACED RELATIVE TO DISPLAYED TEXT. WITHIN THE TEXT FILE, THERE ARE CONTROL COMMANDS SUCH AS GRAPHIC ACTION COMMANDS (SUCH AS DRAW A POINT OR A LINE) THAT ARE IDENTIFIED BY AN ESCAPE CHARACTER AND A TWO LETTER COMMAND IDENTIFIER.

DESIGN APPROACH

IN ORDER TO MAINTAIN AS MUCH FLEXABILITY AS POSSIBLE, A HYBIRD DESIGN APPROACH WAS ADOPTED. FIRST, WITH A BOTTOM-UP DESIGN, THE BASIC ROUTINES THAT EXPRESS THE CAPABILITY OF THE ORION TERMINAL WERE WRITTEN. WITH THESE PROCEDURES WRITTEN, IT WAS THEN POSSIBLE TO USE THEM AS TOOLS TO TEST HIGH-LEVEL, USER-ORIENTED CONCEPTS. ONCE THE HIGH-LEVEL USER INTERFACE WAS CRYSTALIZED, A SWITCH TO TOP-DOWN DESIGN WAS APPECTED.

THIS APPROACH HAD THE ADVANTAGE OF KEEPING THE FLEXABILITY OF THE LOWER LEVEL PROCEDURES FROM CONVERGING TO QUICKLY TO ONE USER ORIENTED INTERFACE. THUS, ONE COULD STRIP OFF THE TOP LEVEL INTERFACE FROM THE DISPLAY MODULE AND STILL HAVE A USEFULL SET OF TOOLS. ONE COULD, FOR INSTANCE, WRITE A TEKTRONIX EMULATOR USING THE LOWEST LEVEL PROCEDURES. AS A RESULT OF THIS, YOU WILL FIND SOME OF THE OPTIONS TO SOME OF THE LOWER LEVEL PROCEDURES NEVER USED IN THE TOP LEVEL DESIGN.

HIGH LEVEL OPERATION

THE USER IS PRESENTED WITH 4 TYPES OF PROCEDURES. ONE SET IS FOR BLOCK MAINTENANCE, ONE SET IS FOR TOUCH PANEL INPUT, ONE PROCEDURE IS FOR TEXT DISPLAY, AND THE LAST PROCEDURE IS USED TO INITIALIZE THE SYSTEM.

THE FRAMEWORK FOR ANYTHING DISPLAYED ON THE SCREEN ARE ALL CURRENTLY CREATED BLOCKS. THEREFORE, IT IS NECESSARY TO CREATE A BLOCK BEFORE WRITING ANYTHING ON THE SCREEN. HERE IS AN EXAMPLE OF A BLOCK CREATION CALL:

```
   CREATEBLOCK(ID,0,512,0,512,DOTS,DOTS,LL,5,NOSCROLL
                WRAP,STANDARD,ALTERNATE,ERROR);
```

THIS CREATES A BLOCK THAT COVERS THE WHOLE SCREEN, HAS A BORDER THICKNESS OF 5 DOTS, END OF PAGE CONDITION CAUSES THE SCREEN TO CLEAR AND THEN THE NEW TEXT WRITTEN, END OF LINE CONDITION CAUSES WRAPPING OF THE TEXT. THE DEFAULT CHARACTER SET IS STANDARD. THE ALTERNATE CHARACTER SET IS THE PROGRAMMABLE ONE. WHEN CALLED, CREATEBLOCK PROVIDES AN ID FOR YOU WHICH YOU MUST PROVIDE TO THE OTHER ROUTINES TO REFER TO THE BLOCK JUST CREATED. ERROR IS SET TO ANY OF THE POSSIBLE BLOCK CREATION ERRORS.

NOW, LET'S SAY THAT YOU WANT TO DISPLAY A QUESTION WHICH THE USER IS EXPECTED TO ANSWER USING THE TOUCH PANEL. YOU FIRST WRITE THE TEXT OF THE QUESTION TO A TEXT FILE FOLLOWED BY THE SET OF TT ESCAPE SEQUENCES, ONE FOR EACH OF THE POSSIBLE ANSWERS. FOR INSTANCE, LET'S SAY YOU WANT TO ASK THE QUESTION: "HOW ARE YOU TODAY?", AND THE TWO POSSIBLE ANSWERS ARE: "HORRIBLE!" AND "FINE." ONE WOULD CREATE A TEXT FILE WITH THE FOLLOWING TEXT IN IT:

```
HOW ARE YOU TODAY?\EL\TT1HORRIBLE!\TE\TT2FINE.\TE\EL
```

THE \EL SEQUENCE REPRESENTS THE END OF LINE DELIMITER. THE \TT SEQUENCE IS THE TOUCH TARGET COMMAND. THE NUMBER FOLLOWING THE TT IS A TT IDENTIFIER WHICH IS USED TO IDENTIFY WHAT TT WAS TOUCHED. LET'S SAY THAT THE TEXT IS IN THE TEXT FILE OUTTEXT. TO DISPLAY IT IN THE BLOCK CREATED ABOVE YOU WOULD DO THE FOLLOWING:

```
   DSTEXT(ID,OUTTEXT,ERROR);
```

ONCE WE PRESENTED THE QUESTION(S), WE ARE READY TO RETRIEVE THE ANSWER. WE DO THE FOLLOWING:

```
   GETTARGINP(BLOCKID,TARGID,CHARRAY,CHLEN,FALSE,ERROR);
```

```
000003   IF  THERE  WAS  A LEGITIMATE RESPONSE FROM THE USER, THEN THIS PROCEDURE
000003   WILL COME BACK WITH A BLOCK IDENTIFIER IN WHICH A TARGET WAS TOUCHED AND
000003   THE  TARGET  IDENTIFIER  THAT WAS PROVIDED IN THE \TT  COMMAND SEQUENCE.
000003   IF ERROR WAS MISTOUCH, THE USER TOUCHED  OUTSIDE  THE  SENSITIVE  TARGET
000003   AREAS.  IF  ERROR  WAS BADTOUCH, THE USER PROBABLY ATTEMPTED TO USE THE
000003   KEYBOARD TO ANSWER THE QUESTION.  THE BUFFER THAT WAS USED TO INPUT  THE
000003   TEXT  IS PROVIDED SO THAT YOU CAN RETRIEVE THE KEYBOARD INPUT.  THE PARA
000003   WERE ENTERED.
000003
000003
000003    THERE IS A SET OF PROCEDURES TO CARRY ON WITH THE BLOCK AFTER  IT  HAS
000003   BEEN  USED.  THESE  ALL  TAKE  TWO ARGUMENTS, A BLOCK IDENTIFIER AND AN
000003   ERROR RETURN.  DSTBLOCK DESTROYS A BLOCK.  CLEARBLOCK AND UNDOBLOCK  ARE
000003   WAYS  OF READYING A BLOCK FOR NEW STUFF.  REDOBLOCK IS FOR WHEN YOU HAVE
000003   TRANSMISSION ERRORS.   DISARMBLOCK  DEACTIVATES  ALL  TTS  IN  A  BLOCK.
000003   REARMBLOCK REARMS THE TTS IN A BLOCK AS WELL AS RESETTING THEM VISUALLY.
000003
000003
000003    TO INITIALIZE THE SYSTEM, DO THE FOLLOWING:
000003
000003      INITDSPARRAYS(CHFILE);
000003
000003   WHERE CHFILE IS OF TYPE ALFA  AND  HAS  AS  A  VALUE  THE  NAME  OF  THE
000003   PROGRAMMALE CHARACTER SET FILE.
000003
000003
000003   ---------------- P R O C E D U R E S ----------------
000003                    ------------------
000003
000003                          U S E D
000003                          -------
000003
000003   PROG# PROCEDURE LENGTH   PROCEDURES CALLED
000003   ----- --------- ------   -----------------
000003
000003    (  1) DISPLAY    565     OPEN       CLOSE      PUTCH      CNVT       CNVTA
000003                             DSPCHARCNT CNTCHARS   DISABLED   NEST       CONVMETRIC
000003                             CENTERIT   MODE       SETCOORD   DRAWLINE   DRAWPOINT
000003                             PUTCHAR    DRAWCHAR   LOADCHRS   GETLN      GETSET
000003                             DRAWBOX    CREATETARG DSTTARG    FETCH      DSPLINE
000003                             GETTOUCHIN HANDLEECL  INITDSPARR CREATEBLOC DSTBLOCK
000003                             CLEARBLOCK UNDOBLOCK  REDOBLOCK  DISARMBLOC REARMBLOCK
000003                             DSTEXT     GETTARGINP
000003
000003                          INTERNAL                          EXTERNAL
000003                          --------                          --------
000003    (  2) OPEN         8
000003    (  3) CLOSE        6
000003    (  4) PUTCH       34                                    [WRITE,WRITELN]
000003    (  5) CNVT        44                                    [READ,ORD]
000003    (  6) CNVTA       44                                    [READ,ORD]
000003    (  7) DSPCHARCNT  41
000003    (  8) CNTCHARS    27                                    [DSPCHARCNT]
000003    (  9) DISABLED    30
000003    ( 10) NEST        48
000003    ( 11) CONVMETRIC  43
000003    ( 12) CENTERIT    49
```

```
000003   443  ( 13) MODE        117      SETUP                                         [PUTCH]
000003   444  ( 14) SETUP        19                                                    [PUTCH]
000003   445  ( 15) SETCOORD     42                                                    [PUTCH,CONVMETRIC,MODE]
000003   446  ( 16) DRAWLINE     48                                                    [PUTCH,CONVMTERIC,MODE]
000003   447  ( 17) DRAWPOINT    35                                                    [PUTCH,CONVMETRIC,MODE]
000003   448  ( 18) PUTCHAR      82                                                    [PUTCH,MODE]
000003   449  ( 19) DRAWCHAR     37                                                    [CONVMETRIC,MODE,PUTCHAR,CENTERIT]
000003   450  ( 20) LOADCHRS     73                                                    [PUTCH]
000003   451  ( 21) GETLN        43                                                    [READ,EOF,EOLN]
000003   452  ( 22) GETSET      205                                                    [GETLN,READLN]
000003   453  ( 23) DRAWBOX      68                                                    [CONVMETRIC,CENTERIT,DRAWPOINT,
000003   454  ( 23) DRAWBOX      68                                                    DRAWLINE,SETCOORD,ABS]
000003   455  ( 24) CREATETARG  108                                                    [CONVMETRIC,CENTERIT,CNTCHARS,
000003   456                                                                           DRAWBOX,SETCOORD,MODE,NEW,
000003   457                                                                           WRITE]
000003   458  ( 25) DSTTARG      15                                                    []
000003   459  ( 26) FETCH        28
000003   460  ( 27) DSPLINE     296                                                    [SETCOORD,FETCH,MODE,
000003   461                                                                           PUTCH,DSPLINE,DRAWLINE,
000003   462                                                                           DRAWPOINT,DRAWCHAR,PUTCHAR,
000003   463                                                                           CREATETARG,DSTARG,WRITE]
000003   464  ( 28) GETTOUCHIN   11                                                    [SETCOORD,GETLN,MODE,
000003   465                                                                           PUTCH,CNVT1,DRAWCHAR,
000003   466                                                                           WRITELN,GETSEG,EOS]
000003   467  ( 29) HANDLEEOL    51                                                    [GETTOUCHINP,DSPLINE,DISPOSE]
000003   468  ( 30) INITDSPABR  155                                                    [LINELIMIT,OPEN,CLOSE,
000003   469                                                                           GETSET,SETCOORD,MODE,
000003   470                                                                           LOADCHRS,WRITE,NEW,
000003   471                                                                           PUTCH,RESET]
000003   472  ( 31) CREATEBLOC  177      STATUS                                        [CONVMETRIC,CENTERIT,DRAWBOX,
000003   473                                                                           SETCOORD,NEW]
000003   474  ( 32) STATUS       58                                                    [NEST]
000003   475  ( 33) DSTBLOCK     70      KILLBLOCK                                     [MODE,PUTCH,SETCOORD,
000003   476                                                                           DRAWLINE,DISABLED]
000003   477  ( 34) KILLBLOCK    52                                                    [KILLBLOCK,DISPOSE]
000003   478  ( 35) CLEARBLOCK   11                                                    []
000003   479  ( 36) UNDOBLOCK    36                                                    [DSPLINE,DISPOSE]
000003   480  ( 37) REDOBLOCK    11                                                    []
000003   481  ( 38) DISARMBLOC   25
000003   482  ( 39) REARMBLOCK   27
000003   483  ( 40) DSTEXT      122      CNVT1        PUTICH       INTERPRET         [CNVT,READ,READLN,
000003   484                                                                           EOLN,EOF,DISPOSE]
000003   485  ( 41) CNVT1        43                                                    [READ,CNVT]
000003   486  ( 42) PUTICH       44                                                    [NEW]
000003   487  ( 43) INTERPRET   123                                                    [PUTICH,INTERPRET,HANDLEEOL,
000003   488                                                                           DSPLINE,NEW,READ,
000003   489                                                                           REWRITE,WRITELN,RESET,EOLNS]
000003   490  ( 44) GETTARGINP   84                                                    [GETTOUCHINP,DISABLED,DRAWBOX,
000003   491                                                                           SETCOORD,MODE,WRITE]
000003   492
000003   493
000003   494 ---------------- R E F E R E N C E S ----------------
000003   495                  ------------------
000003   496     A DESCRIPTION OF AN EDITOR USING THE CRICN-60 WITH
000003   497 A TOUCH PANEL IS FOUND IN MITCHELL LUBARS'S M.S. THESIS:
000003   498        AN EDITOR/GENERATOR SYSTEM FOR CREATICN OF
000003   499        VARIABLE PHRASE KEYBOARDS. 1980.
```

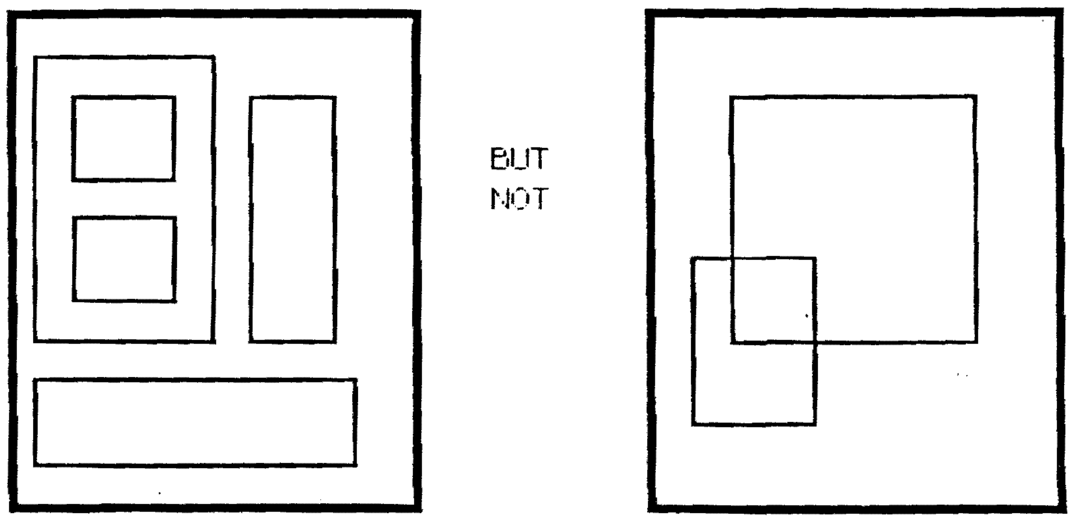
```
000003    500
000003    501     A DESCRIPTION OF A TERMINAL DRIVER SYSTEM THAT HANDELED
000003    502 ORION-60 TERMINALS IS FOUND IN FLINT PELLETT'S M.S. THESIS:
000003    503          TERMINAL MANAGEMENT FOR A USER-ORIENTED SYSTEM.
000003    504          1979.
000003    505
000003    506      THE CENTER FOR ADVANCED COMPUTATION DOCUMENT THAT
000003    507 DESCRIBES THE INTELLIGENT TERMIMANL SOFTWARE WHICH WAS
000003    508 A LARGE DISPLAY PACKAGE WRITTEN IN C USING A PLASMA
000003    509 TERMINAL WITH TOUCH PANEL IS:
000003    510
000003    511          INTELLIGENT TERMINAL PROGRAMMER'S MANUAL
000003    512                         BY
000003    513          BROWN, KOPETZKY, MULLEN, & WILLCOX
000003    514          IN 2 VOLUMES, CAC DOCUMENT # 236. OCT 31 1977.
000003    515 THIS REFERENCE IS WHERE I GOT THE TERM "TOUCH TARGET".
000003    516
000003    517     THE MANUALS THAT DESCRIBE THE ORION-60 & THE TOUCH PANEL
000003    518 FOR THE ORION-60:
000003    519
000003    520          ORION-60 PLASMA DISPLAY TERMINAL
000003    521          INSTALLATION & OPERATION MANUAL.
000003    522
000003    523                         &
000003    524
000003    525          TOUCH INPUT SYSTEM OPTION USERS MANUAL
000003    526                         FOR
000003    527          MODEL 12,000 PLASMA DISPLAY TERMINAL (MODEL 27)
000003    528                         &
000003    529          ORION-60 PLASMA DISPLAY TERMINAL (MODEL 28).
000003    530
000003    531     ALSO, IF THE ORION IS SICK, REFER TO:
000003    532
000003    533          ORION-60 PLASMA DISPLAY TERMINAL
000003    534          MAINTENANCE DOCUMENTATION.
000003    535
000003    536      THE ADDRESS FOR MAGNAVOX IS:
000003    537
000003    538           MAGNAVOX GOVERMENT & INDUSTRIAL ELECTRONICS CO.
000003    539           1313 PRODUCTION ROAD
000003    540           FORT WANE, INDIANA 46808
000003    541           219 482-4411
000003    542
000003    543      THE ABOVE THESES AND CAC DOCUMENTS ARE IN THE
000003    544 DCL LIBRARY. THE ORION DOCUMENTS SHOULD BE EITHER
000003    545 WITH ME OR WITH THE ORION. A.B. BASKIN ALSO HAS
000003    546 COPIES OF SOME OF THE ORION DOCUMENTS.
000003    547
000003    548
000003    549     THERE IS A SET OF SCHEMATICS ON THE CARROLL TOUCH
000003    550 PANEL THAT SHOULD BE ASSOCIATED WITH THE ORION-60
000003    551 MAINTENANCE DOCUMENT. THIS PANEL IS A 32X32 THAT BELONGED
000003    552 TO THE INTELLIGENT TERMINAL SYSTEM (THE LSI-11 IN THE QUME ROOM)
000003    553 I KLUDGED IT SO IT WOULD WORK WITH THE ORION. THERE IS
000003    554 A SWITCH-YARD BOARD LOCATED IN ONE OF THE ACCESSORY CARD
000003    555 SLOTS THAT CONDUCTS THE KLUDGE. I DROP THE LSB.
000003    556 ------------------------------------------------------
```

9

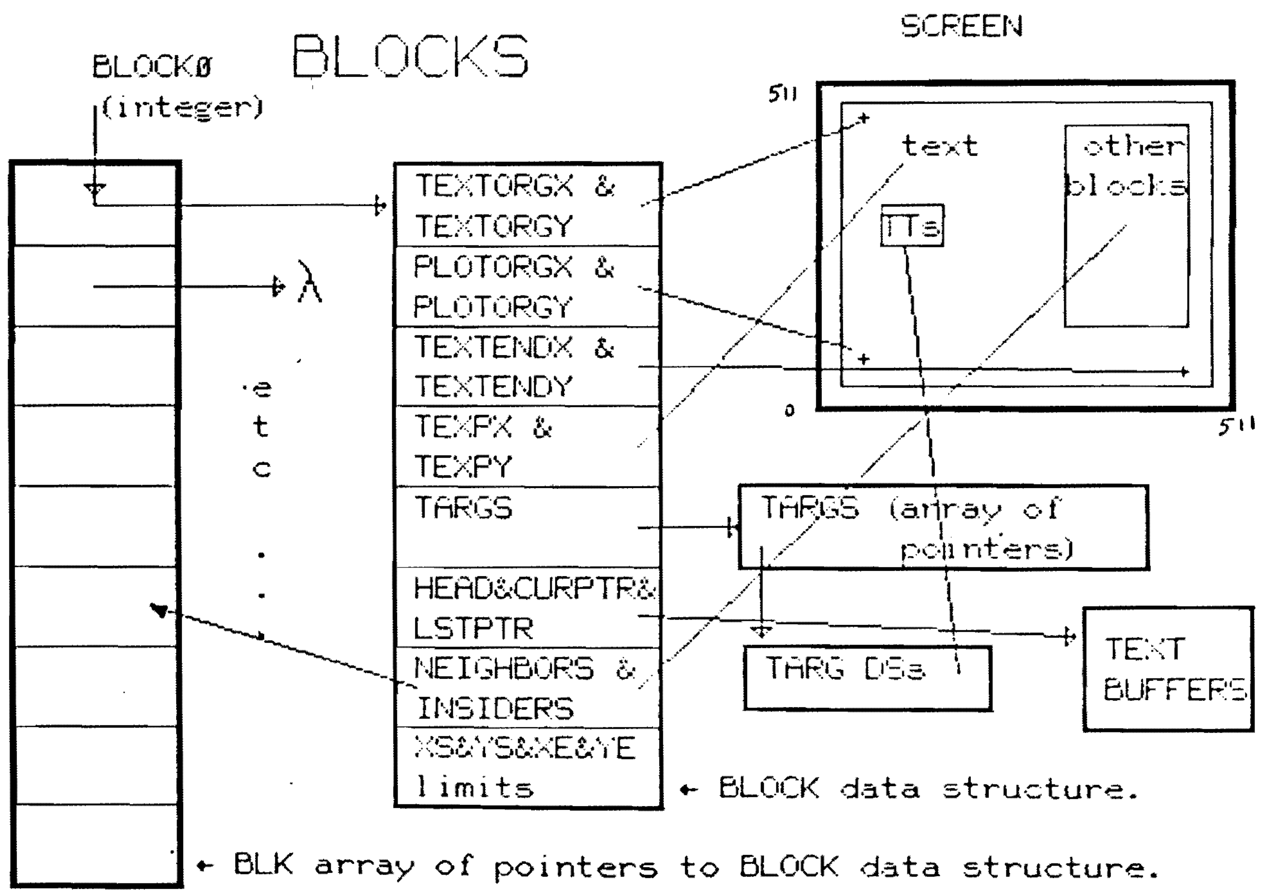
```
000003    557 *)
000003    558
000003    559
000003    560
000003    561
```

Proper Nesting of Blocks

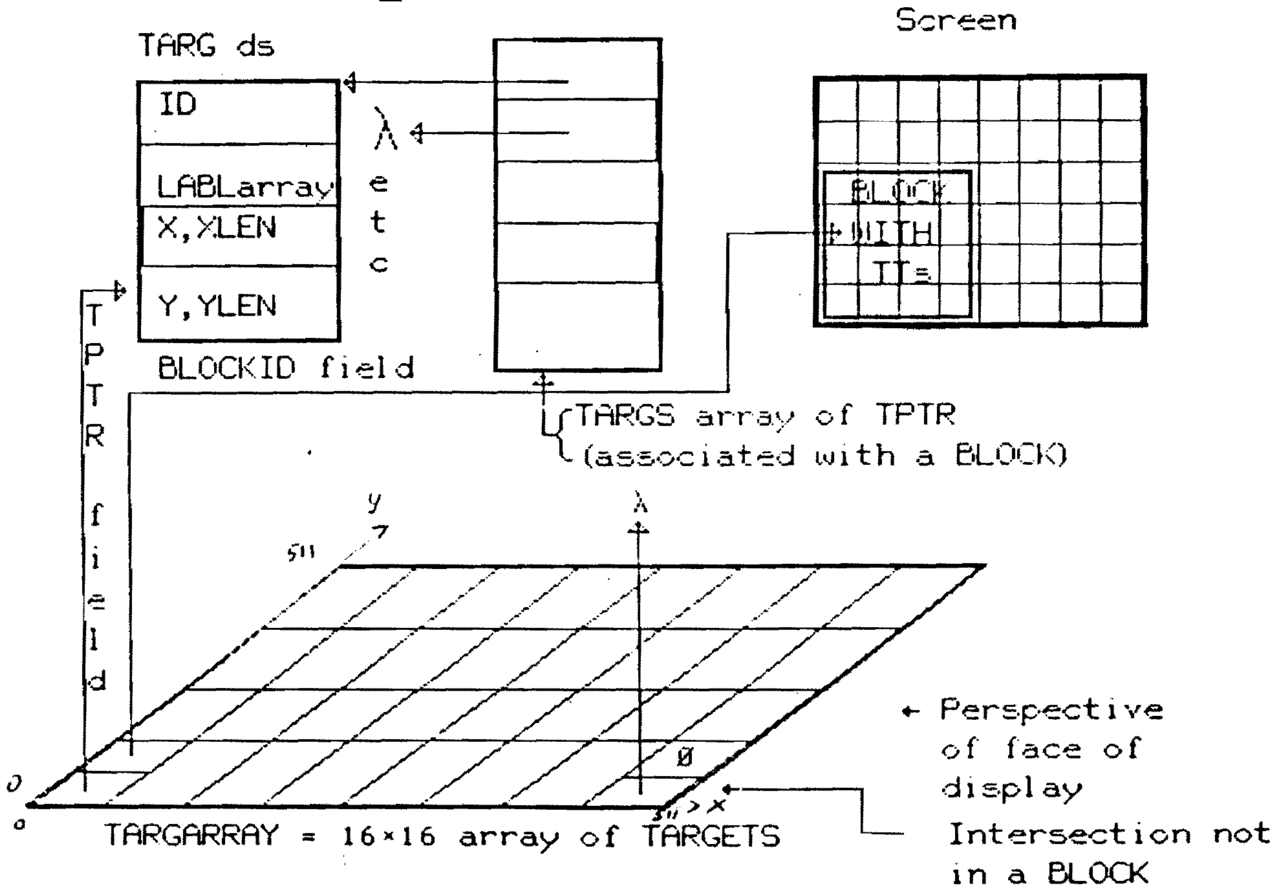
Blocks can be nested but not overlapped



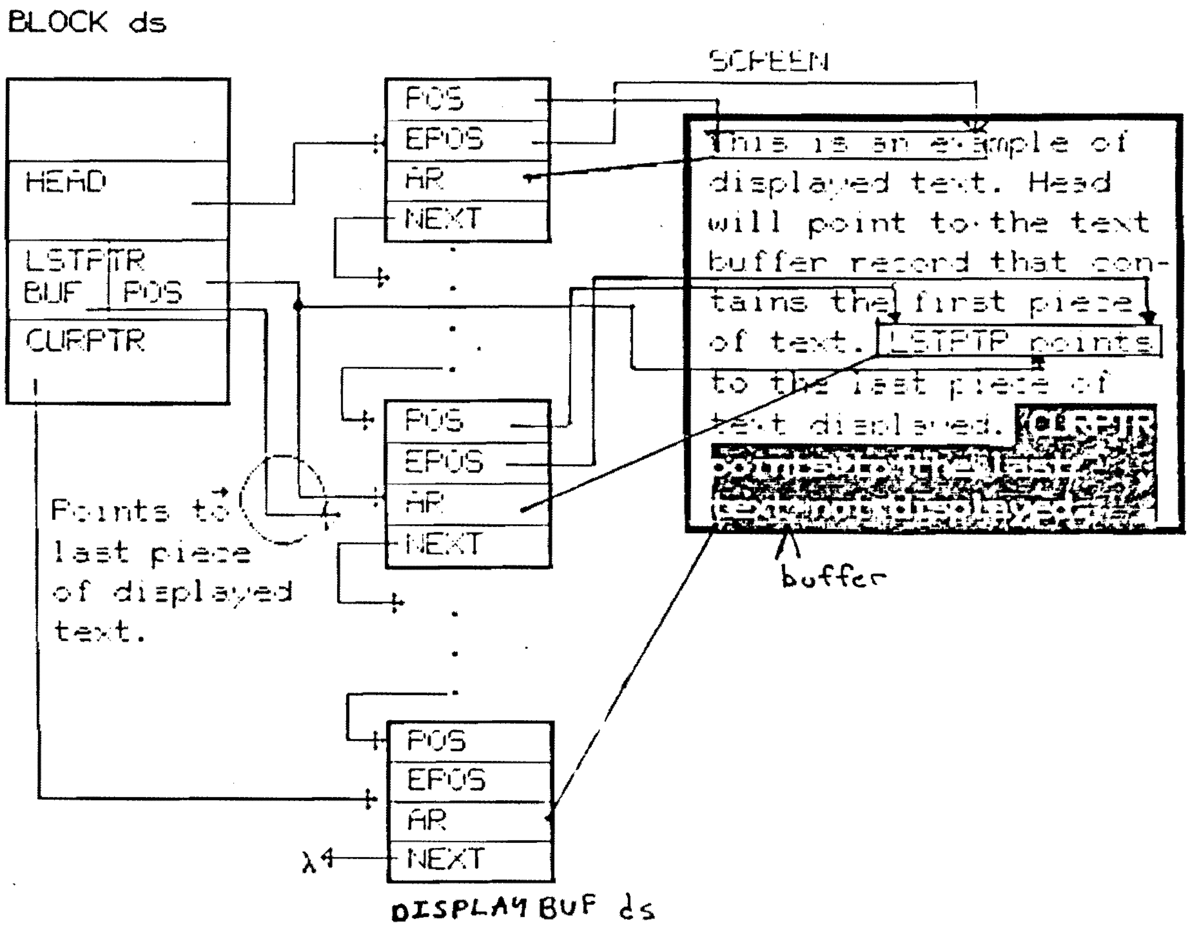
DATA STRUCTURES FOR BLOCKS



Data Structures for Targets


TARG ds
ID
λ
etc
LABLarray
X,XLEN
Y,YLEN
BLOCKID field
TPTR field
Screen
BLOCK WITH TT=
TARGS array of TPTR (associated with a BLOCK)
y
λ
5"
0
0
3"
x
Ø
TARGARRAY = 16×16 array of TARGETS
← Perspective of face of display
Intersection not in a BLOCK

Data Structures for Text



BLOCK ds
HEAD
LSTPTR
BUF
POS
CURPTR
SCREEN
POS
EPOS
AR
NEXT
This is an example of displayed text. Head will point to the text buffer record that contains the first piece of text. LSTPTR points to the last piece of text displayed.
buffer
Points to last piece of displayed text.
λ
DISPLAY BUF ds

```
000004       2 (*$L'DATA TYPES FOR DISPLAY MODULE' *)
000004       3 (*$I'GLOBAL'/'KBLTB'*)

------ BEGIN INCLUDED TEXT.

000004       3 PROGRAM KBMAIN(KB,INPUT/+,OUTPUT,MSG);
000074       3
000074       3                                                   (* GLOBAL CONSTANTS *)
000074       3
000074       3 CONST
000074       3       CONMAXPRINTNAME   = 256;                    (*MAX LENGTH OF PRINTNAME*)
000074       3       CONMAXTUPLELENGTH = 256;                    (* MAX TUPLE LENGTH *)
000074       3
000074       3       (* DISPLAY MODULE CONSTANTS *)
000074       3       BUFLEN = 132; (* LENGTH OF INPUT BUFFERS USED IN DISPLAY MODULE *)
000074       3       ESCAPE = '\'; (* DISPLAY MODULE ESCAPE CHARACTER *)
000074       3
000074       3                                                   (* GLOBAL TYPE DEFINITIONS *)
000074       3
000074       3
000074       3 TYPE
000074       3       DUMMY             = INTEGER;                (* DUMMY STATIC AREA - DO NOT USE *)
000074       3
000074       3       INTNAMEPTR        = ^INTNAMEREC;            (*POINTERS TO INTERNAL NAME RECORDS*)
000074       3       INTNAMEFLAG       = (NODEFLAG, INTFLAG, REALFLAG);
000074       3                                                   (*FLAG IN INTERNAL NAME INDICATING
000074       3               WHETHER NAME IS AN ACTUAL NODE OR AN INTEGER OR REAL VALUE*)
000074       3
000074       3
000074       3 (**************************************************************
000074       3      THE FOLLOWING IS THE STRUCTURE OF THE INTERNAL NAME RECORD
000074       3  TYPE.  THIS STRUCTURE IS INVISIBLE TO THE USER, HOWEVER.  THE
000074       3  USER DEALS ONLY WITH THESE RECORDS AS A WHOLE, AND ONLY THE
000074       3  FOLLOWING SUBROUTINE PRIMITIVES ACCESS INDIVIDUAL FIELDS.
000074       3   USER OPERATIONS UPON INTERNAL NAMES ARE RESTRICTED TO ASSIGNMENTS
000074       3    OF THE TYPE A:=B , THE USER SUPPLIED FUNCTION EQ TO TEST FOR
000074       3    EQUIVALENCE (E.G. IF (EQ(A)=TRUE THEN ...), AND THE PROCEDURES
000074       3    INTERINT AND INTERREAL FOR CONVERTING BACK AND FORTH FROM
000074       3    INTEGER AND REAL NUMBERS TO INTERNAL NAMES.
000074       3 ***************************************************************)
000074       3       INTNAMEREC        = RECORD                  (*AN INTERNAL NAME CONSISTS OF A FLAG*)
000074       3                             CASE NF: INTNAMEFLAG OF
000074       3                                                   (*AND A MODIFIER DEPENDANT UPON THE FLAG*)
000074       3 NODEFLAG:
000074       3          (NODE             : INTEGER);            (*ACTUAL NODE*)
000074       3 INTFLAG:
000074       3          (IVALUE           : INTEGER);            (*INTEGER VALUE*)
000074       3 REALFLAG:
000074       3          (RVALUE           : REAL);               (*REAL VALUE*)
000074       3                             END;
000074       3
000074       3
000074       3       LEVELTYPE         = (PRIVATELEV,            (*PRIVATE LEVEL ACCESS TO NODE*)
000074       3       GROUPLEV,                                   (*GROUP LEVEL *)
000074       3       GLOBALLEV);                                 (*GLOBAL LEVEL *)
000074       3
000074       3
```

```
000074      3       PRINTNAME          = PACKED ARRAY [1 .. CONMAXPRINTNAME ] OF
000074      3                         CHAR;                      (*CHAR STRING WHERE PRINTNAME IS STORED*)
000074      3       NODEACCESSTYPE     = (RDONLY, RDWRT);
000074      3                                                    (*NODE ACCESS IS READ ONLY OR READ/WRITE*)
000074      3       TUPLE              = ARRAY [1 .. CONMAXTUPLELENGTH] OF
000074      3                         INTNAMEREC;
000074      3                                                    (*ARRAY TYPE WHERE INTERNAL NAME POINTERS OF TUPLE ARE STORED*)
000074      3       TUPLELENGTH        = 0 .. CONMAXTUPLELENGTH;
000074      3                                                    (*A NUMBER TO INDICATED HOW MANY MEMBERS IN A TUPLE*)
000074      3       PRINTNAMELENGTH    = 0 .. CONMAXPRINTNAME;  (*NUMBER OF CHARS IN PRINTNAME
000074      3                                      NOT COUNTING THE EOL*)
000074      3
000074      3       ERRORTYPE          = (NOERROR,               (*NO ERROR OCCURRED*)
000074      3                            ERRMAXNODES,            (*MAXIMUM NUMBER OF NODES USED*)
000074      3                            ERRMAINNTFND,           (*MAIN NODE WAS NOT FOUND*)
000074      3                            ERRCNTNOTO,             (*USAGE OR ATTR. COUNT OF NODE WAS NOT 0*)
000074      3                            ERRMAXATTR,             (*NODE HAS MAX NUMBER OF ATTRIBUTES*)
000074      3                            ERRMAXUSAGE,            (*NODE'S USAGE COUNT HAS REACHED MAX*)
000074      3                            ERRREADONLY,            (*WRITE ACCESS REQUESTED TO RD ONLY*)
000074      3                            ERRPRINTNAME,           (*PRINTNAME NOT EVALUATABLE*)
000074      3                            ERRTOASCII,             (*ERROR IN CONVERING NUMBER TO ASCII*)
000074      3                            ERRFROMASCII,           (*ERROR IN CONVERING FROM ASCII *)
000074      3                            ERRINTNODE,             (*INTERNAL NAME WAS NOT A NODE*)
000074      3                            ERRLOOKUPFAIL,          (*PRINTNAME NOT IN DICTIONARY*)
000074      3                            ERRDICTFUL,             (*DICTIONARY FULL*)
000074      3                            ERRTUPNTFND,            (*TUPLE WAS NOT FOUND*)
000074      3                            ERROPEOF,               (*EOF ENCOUNTERED IN OPENNETWORK*)
000074      3                            ERROPMSCH,              (*MISSING CHARACTER*)
000074      3                            ERROPCREATE,            (*OPENNETWORK ERROR FROM CREATENODE*)
000074      3                            ERROPSETPRI,            (*     "      "     "  SETPRINTNAME*)
000074      3                            ERROPENTER,             (*     "      "     "  ENTERDICT*)
000074      3                            ERROPLOOKUP,            (*     "      "     "  LOOKUP*)
000074      3                            ERROPADDATT,            (*     "      "     "  ADDATTRIBUTE*)
000074      3                            ERROPOPEN,              (*     "      "     "  OPENNODE*)
000074      3                            ERROPCLOSE,             (*     "      "     "  CLOSENODE*)
000074      3                            ERRCLOGETPRI,           (*CLOSENETWORK "     "  GETPRINTNAME*)
000074      3                            ERRCLOGETATT,           (*     "       "     "  GETATTRIBUTE*)
000074      3                            ERRINTERINT,            (*INTERNAL NAME WAS NOT AN INTEGER*)
000074      3                            ERRINTERREAL,           (*INTERNAL NAME WAS NOT A REAL*)
000074      3                            ERRPRSYNTAX,            (*PARSER FOUND A SYNTAX ERROR*)
000074      3                            ERRPROVERFLOW,          (*PARSER STACK OVERFLOW*)
000074      3                            ERRPRNOPARTAB,          (*PARSER TABLES FILE MISSING*)
000074      3                            ERRPRINTERNAL,        (*PARSER INTERNAL ERROR *)
000074      3                           (*--- DISPLAY MODULE ERRORS ---*)
000074      3                            BLOCKNOTNESTED,       (*DISPLAY BLOCK NOT NESTED RIGHT *)
000074      3                            TOOMANYBLOCKS,        (*TRIED TO MAKE MORE THAN BLOCKMAX BLOCKS *)
000074      3                            BLOCKOFFSCREEN,       (*TRIED TO CREATE A BLOCK THAT GOES OFF THE SCREEN *)
000074      3                            NOSUCHBLOCK,          (*THE BLOCK WAS NEVER CREATED *)
000074      3                            OUTOFBOUNDS,          (*TRIED TO PUT SOMETHING OFF THE SCREEN *)
000074      3                            NOMEMORY,             (* RAN OUT OF INTERNAL MEMORY IN DSPTEXT *)
000074      3                            COMMANDERROR,         (*COMMAND SYNATX ERROR IN INPUT FILE TO DSPTEXT *)
000074      3                            MISTOUCH,             (*AN ACTIVE TOUCH TARGET WAS NOT TOUCHED *)
000074      3                            BADTOUCH,             (*THE USER PROBABLY TYPED IN A RESPONSE TO TT REQUEST *)
000074      3                            LABELTOOLONG          (*A TOUCH TARGET LABEL WAS TOO LONG *)
000074      3                          );
000074      3
000074      3
```

```
000074                                    (* PARSER TUPLE MARKERS *)
000074        MARKTYPE          = (MKNULL,MKDOMAIN,MKNOMINAL,MKINTERVAL,MKSTRUCTURE,
000074                             MKUNITS,MKREFINE,MKTRUE,MKFALSE,MKREF0,MKREF1,MKREF2,
000074                             MKVAR,MKFUNC1,MKFUNC2,MKEQ,MKGT,MKLT,MKNE,MKGE,MKLE,
000074                             MKALL,MKUNKN,MKUNDEF,MKNA,MKBEHAVIOR,MKINCR,MKDECR,
000074                             MKRMAX,MKRMIN,MKARGS,MKRULES,MKVARS,MKVARDCL,MKRULE,
000074                             MKPAREN,MKCS,MKAND1,MKAND2,MKOR1,MKOR2,MKEXCPT,MKEQUIV,
000074                             MKIMP,MKLM);
000074
000074        (*------ T Y P E S   F O R   D I S P L A Y   M O D U L E -----*)
000074        METRIC = (DOTS,CHS,PARTS); (* METRIC FOR SCREEN COORDS FOR DISPLAY MODULE *)
000074        INBUF = PACKED ARRAY[1..BUFLEN] OF CHAR; (* INPUT BUFFER USED IN DISPLAY MODULE *)
000074        SCROLLTYPE = (SCROLL,NOSCROLL); (*TYPE SPECIFICATION FOR SCROLLING FOR DISPLAY MODULE *)
000074        WRAPTYPE = (WRAP,TRUNCATE); (*EOL SPECIFICATION FOR DISPLAY MODULE*)
000074        CHSETTYPE = (STANDARD,ALTERNATE); (*CHARACTER SET SPECIFICATION FOR DISPLAY MODULE *)
000074        ADJ = (CENTER,LL,UL,LR,UR) ; (* ORIGIN SPECIFICATION FOR DISPLAY MODULE *)
000074
000074

------ END INCLUDED TEXT.

000074      CONST
000074         STSPEC = 128;
000074         ESCCH = 92;     LN = 129;   PT = 130;
000074         CH = 131;   CA = 132;   SC = 133;
000074         ML = 134;   MR = 135;   MU = 136;
000074         MB = 137;   PL = 138;   OV = 139;
000074         OA = 140;   UN = 141;   UE = 142;
000074         NP = 143;   ND = 144;   RP = 145;
000074         RD = 146;   CR = 147;   LI = 148;
000074         AL = 149;   NR = 150;   TT = 151;
000074         TU = 152;   TE = 153;   GO = 154;
000074         SU = 155;   SA = 156;   UU = 157;
000074         UA = 158;
000074
000074       (* ASCII CONSTANTS *)
000074         NUL = 0;
000074         SOH = 1;
000074         STX = 2;
000074         ETX = 3;
000074         EOT = 4;
000074         ENQ = 5;
000074         ACK = 6;
000074         BEL = 7;
000074         BS = 8;
000074         HT = 9;
000074         LF = 10;
000074         VT = 11;
000074         FF = 12;
000074         DSPCR = 13;
000074         SO = 14;
000074         SI = 15;
000074         DLE = 16;
000074         DC1 = 17;
000074         DC2 = 18;
000074         DC3 = 19;
000074         DC4 = 20;
```

```
000074          NAK = 21;
000074          SYN = 22;
000074          ETB = 23;
000074          CAN = 24;
000074          EM = 25;
000074          SUB = 26;
000074          ESC = 27;
000074          DSPFS = 28;
000074          GS = 29;
000074          RS = 30;
000074          US = 31;
000074          DELETE = 127;
000074          BLANK = 32;
000074  (*PSEUDONAMES*)
000074          KEYLOCK = 2; (* STX *)
000074          ERASE = 3; (* ETX *)
000074          DSPNORM = 4; (* EOT *)
000074          OVER = 6; (* ACK *)
000074          INVERSE = 16; (* DLE *)
000074          BACK = 8; (* BS *)
000074          FOWARD = 9; (* HT *)
000074          UP = 11; (* VT *)
000074          HOME = 12; (* FF *)
000074          CLEAR = 19; (* DC3 *)
000074          SUBSCRIPT = 18; (* DC2 *)
000074          SUPERSCRIPT = 28; (* FS *)
000074          ALTCHAR = 14; (* SO *)
000074          NORCHAR = 15; (* SI *)
000074          TCH = 5; (* ENQ *)
000074          BHOME = 30; (* RS *)
000074          DSPYES = 1; (*FOR PAGING *)
000074          DSPNO = 0;
000074          (*BUFLEN = 132;*) (* LENGTH OF TEXT BUFFERS *)
000074          LABELLEN = 64; (* LENGTH OF TARG LABELS *)
000074          TEXBUFLEN = 64; (* LENGTH OF DISPLAY TEXT BUFFERS *)
000074          TEXBUFLEN1 = 65; (* TEXBUFLEN + 1 *)
000074          TARGMAX = 20;  (* MAX TARGS/BLOCK *)
000074          BLOCKMAX = 20; (* MAX # OF BLOCKS *)
000074          NCOMMANDS = 30; (* NUMBER OF DSPTEXT COMMANDS *)
000074          (*ESCAPE = '\';*) (* ESCAPE CHARACTER FOR DISPLAY ROUTINES *)
000074      TYPE
000074          CRNG = 0..999;
000074          NESTTYPE = (NESTED,REVNESTED,NOTNESTED,OVERLAP);
000074          (*ERRORTYPE = (
000074                          BLOCKNOTNESTED,
000074                          TOOMANYBLOCKS,
000074                          BLOCKOFFSCREEN,
000074                          NOSUCHBLOCK,
000074                          OUTOFBOUNDS,
000074                          NOMEMORY,
000074                          COMMANDERROR,
000074                          MISTOUCH,
000074                          BADTOUCH,
000074                          LABELTOOLONG
000074                          ); *)
000074          LABELS = PACKED ARRAY[1..LABELLEN] OF CHAR;
000074          LABPTR = ^LABELS; (* A POINTER FOR TARGET LABELS *)
```

```
000074      LINEMODE = (PRINT,UNDO,CONSUME); (* MODES OF DISPLAYING A LINE *)
000074      ARR = PACKED ARRAY[1..TEXBUFLEN] OF 0..511;
000074      DSPBUFPTR = ^DISPLAYBUF;
000074      DISPLAYBUF = RECORD
000074          AR : ARR; (* TEXT BUFFER *)
000074          POS : 0..TEXBUFLEN; (* BEGINNING OF GOOD TEXT IN BUFFER *)
000074          EPOS : 0..TEXBUFLEN1; (* END OF GOOD TEXT IN BUFFER *)
000074          NEXT : DSPBUFPTR; (* NEXT BUFFER *)
000074      END;
000074      TEXTPTR = RECORD (* 2 TUPLE POINTER STRUCTURE FOR POINTING TO TEXT *)
000074          BUF : DSPBUFPTR;
000074          POS : 0..TEXBUFLEN1;
000074      END;
000074      TPTR = ^TARG;
000074      TARGITS = RECORD
000074          BLOCKID : 0..BLOCKMAX;
000074          TARG : TPTR; (* POINTER TO THE TARGET DS *)
000074      END;
000074      (*METRIC = (DOTS,CHS,PARTS);*) (* METRIC FOR SCREEN COORDS *)
000074      TARGTYPE = (PAT,UNDERLINE);
000074      TARGPTR = ^TARGS;
000074      (*INBUF = PACKED ARRAY [1..BUFLEN] OF CHAR; *)
000074      PASBUF = PACKED ARRAY[INTEGER] OF CHAR;
000074      TARG = RECORD
000074          ID : INTEGER;
000074          STYLE : TARGTYPE;
000074          LABL : LABELS;
000074          LBLEN : INTEGER;
000074          X,XLEN : INTEGER;
000074          Y,YLEN : INTEGER;
000074          LORGX,LORGY : INTEGER;
000074          TOUCHED : BOOLEAN;
000074      END;
000074      TARGS = ARRAY[1..TARGMAX] OF TPTR;
000074      (*SCROLLTYPE = (SCROLL,NOSCROLL); *)
000074      (*WRAPTYPE = (WRAP,TRUNCATE); *)
000074      (*CHSETTYPE = (STANDARD,ALTERNATE); *)
000074      BLOCK = RECORD
000074          XS : INTEGER;
000074          XE : INTEGER;
000074          YS : INTEGER;
000074          YE : INTEGER;
000074          OUTLINE : INTEGER;
000074          OVERYFLOW : SCROLLTYPE;
000074          OVERXFLOW : WRAPTYPE;
000074          CHSET : CHSETTYPE;
000074          ALTCHSET : CHSETTYPE;
000074          TEXTORGX : INTEGER;
000074          TEXTORGY : INTEGER;
000074          TEXPX : INTEGER;
000074          TEXPY : INTEGER;
000074          PLOTORGX : INTEGER;
000074          PLOTORGY : INTEGER;
000074          WMODE : INTEGER; (* 3,4,6,16 (ERASE,DSPNORM,OVER,INVERSE) *)
000074          TARGPX : INTEGER;
000074          TEXTENDX : INTEGER;
000074          TEXTENDY : INTEGER;
```

```
000074              LINELEN : INTEGER;
000074              MAXLINE : INTEGER;
000074              CURLINE : INTEGER;
000074              TARGS : TARGPTR;
000074              HEAD : DSPBUFPTR; (* FIRST LINE BUFFER *)
000074              CURPTR : DSPBUFPTR; (* CURRENT LINE BUFFER *)
000074              LSTPTR : TEXTPTR; (* WHERE DISPLAY LEFT OFF *)
000074              INUSE : BOOLEAN;
000074              NEIGHBORS : INTEGER;
000074              INSIDERS : INTEGER;
000074          END;
000074          (*ADJ = (CENTER,LL,UL,LR,UR); *)
000074          ROM = PACKED ARRAY[0..127,0..7] OF 0..177777B;
000074          MAJORSTATE = (DSPTEXT,GRAPHICS);
000074          MINORSTATE = (DSPNONE,TOUCH,PLACE,POINT,LINE,CHPLOT,CHLOAD);
000074          MEMORY = (NORMAL,PROGRAMMABLE);
000074
000074     (* THESE WOULD NORMALLY BE IN THE VARS *)
000074          DSSTATIC = RECORD
000074              CHARSET : ROM; (* CHARCTER SET ARRAY *)
000074              LABARR : ARRAY[1..TARGMAX] OF LABPTR; (* FOR STORAGE OF LABELS BEFORE GIVING THEM TO CREATETARG *)
000074                                                    (* THIS IS SCRAPPED IN DSTBLOCK *)
000074              LABCTR : 0..TARGMAX; (* COUNTER OF NUMBER OF TARGETS *)
000074              LOOKUP : ARRAY[1..NCOMMANDS,1..3] OF CRNG;
000074              DISARM : ARRAY[1..BLOCKMAX] OF INTEGER;
000074              BLOCKO : INTEGER;
000074              SAVEMODE : INTEGER;
000074              BLK : ARRAY[1..BLOCKMAX] OF BLOCK;
000074              TARGARRAY : ARRAY[0..15,0..15] OF TARGETS;
000074              BUILD : ARRAY[0..15,0..7] OF CHAR;
000074              REV : ARRAY[0..15] OF INTEGER;
000074              CHARS : ARRAY[32..127] OF CHAR;
000074              CONV : ARRAY['+'..';'] OF INTEGER;
000074              SAVE1 : MAJORSTATE;
000074              SAVE2 : MINORSTATE;
000074              I,J,K : INTEGER; (* TEMP *)
000074              CBUF : INBUF;
000074          END;
000074          TMSTATIC = DUMMY;
000074          PRSTATIC = DUMMY;
000074          RESTATIC = DUMMY;
000074          S1STATIC = DUMMY;
000074          S2STATIC = DUMMY;
000074     (*$I'GBLVARS'/'KBLIB'*)

------ BEGIN INCLUDED TEXT.

000074     VAR                          (* GLOBAL VARIABLE DEFINITIONS *)
000074
000074         (* POINTERS TO PRIVATE STATIC AREAS *)
000074
000074          KB     : TEXT;          (* KNOWLEDGE BASE FILE *)
000130          MSG    : TEXT;          (* MESSAGES WORK FILE *)
000164          TMSPTR : ^TMSTATIC;     (* FOR THE TUPLE MANAGER *)
000165          PRSPTR : ^PRSTATIC;     (* FOR THE PARSER *)
000166          RESPTR : ^RESTATIC;     (* FOR THE RULE EVALUATOR *)
000167          DSSPTR : ^DSSTATIC;     (* FOR TERMINAL DISPLAY FUNCTIONS *)
```

```
000170          S1SPTR : ^S1STATIC;      (* SPARE 1 *)
000171          S2SPTR : ^S2STATIC;      (* SPARE 2 *)
000172          GBLERROR: ERRORTYPE;     (* MAIN ERROR INDICATOR *)
000173
000173     (*$E+ ASK FOR REAL EXTERNAL NAMES *)
000173     (*$R- WITHOUT REDUCE *)

------ END INCLUDED TEXT.

000173
000173
000173     (*$I'IOAIDS'*)

------ BEGIN INCLUDED TEXT.

000173     (* --- IOAIDS - INPUT/OUTPUT AIDS. --- *)
000173     (*$A+ ASCII CHARACTER SET *)
000173     PROCEDURE SKIPBLANKS(VAR F : TEXT);
000003     BEGIN (* SKIPBLANKS *)
000003     WHILE (F^ = ' ') AND NOT EOF(F) DO GET(F)
000014     END (* SKIPBLANKS *);
000017
000017     FUNCTION EOLNS(VAR F : TEXT) : BOOLEAN;
000004     BEGIN (* EOLNS *)
000004     WHILE (F^ = ' ') AND NOT EOLN(F) DO GET(F);
000015     EOLNS := EOLN(F)
000016     END (* EOLNS *);
000022
000022     FUNCTION EOSS(VAR F : SEGTEXT) : BOOLEAN;
000004     BEGIN (* EOSS *)
000004     WHILE (F^ = ' ') AND NOT EOS(F) DO GET(F);
000015     EOSS := EOS(F)
000016     END (* EOSS *);
000022
000022     FUNCTION EOFS(VAR F : TEXT) : BOOLEAN;
000004     BEGIN (* EOFS *)
000004     WHILE (F^ = ' ') AND NOT EOF(F) DO GET(F);
000015     EOFS := EOF(F)
000016     END (* EOFS *);
000022
000022     PROCEDURE READSTRING(VAR F : TEXT; VAR S : DYNAMIC ALFA;
000004                          LEN : INTEGER);
000011     VAR I,LIM : INTEGER;
000013     BEGIN (* READSTRING *)
000013     IF EOF(F) THEN HALT(' READ PAST EOS/EOF.');
000014     IF LEN < 0
000014      THEN BEGIN SKIPBLANKS(F);
000017       I := 0;
000021       WHILE F^ <> ' ' DO
000025        BEGIN IF I < HIGH(S)
000026         THEN BEGIN I := I + 1; S[I] := F^ END;
000045        GET(F)
000051        END;
000052       LEN := I
000052       END
000054      ELSE BEGIN IF LEN > HIGH(S) THEN LIM := HIGH(S) ELSE LIM := LEN;
000063       FOR I := 1 TO LEN DO
```

```
000066        BEGIN S[I] := F^; IF NOT EOLN(F) THEN GET(F) END;
000117      END;
000117    IF LEN < HIGH(S)
000121     THEN FOR I := LEN+1 TO HIGH(S) DO S[I] := ' '
000136     ELSE FOR I := HIGH(S)+1 TO LEN DO
000147        IF NOT EOLN(F) THEN GET(F)
000156    END (* READS *);
000200    (*$A=*)

------ END INCLUDED TEXT.

000200        PROCEDURE OPEN (
000002                          VAR F : TEXT; (* PASCAL TEXT FILE NAME TO OPEN *)
000003                          N : ALFA; (* LITERAL FILE NAME TO BE OPEN *)
000004                          OPENWRITE : BOOLEAN (* TRUE FOR WRITE *)
000005                      );
000005        EXTERN;
000005
000005        PROCEDURE CLOSE (
000002                          VAR F : TEXT (* PASCAL TEXT NAME TO CLOSE *)
000003                        );
000003        EXTERN;
000003
000003
000003
```

```
000003                                 (*$L'LOW LEVEL ROUTINES'*)
000003                 (*--------------------------------------------------------
000003                 =                                                       =
000003                 =                                                       =
000003                 =                                                       =
000003                 =                                                       =
000003                 =            LL        00000   WW   WW                  =
000003                 =            LL       00   00  WW   WW                  =
000003                 =            LL       00   00  WW   WW                  =
000003                 =            LL       00   00  WW   WW                  =
000003                 =            LL       00   00  WW W WW                  =
000003                 =            LL       00   00  WWWWWWW                  =
000003                 =            LLLLLLL   00000    WW WW                   =
000003                 =                                                       =
000003                 =                                                       =
000003                 =                                                       =
000003                 =  LL        EEEEEEE  V      V EEEEEEE  LL              =
000003                 =  LL        EE        V    V  EE       LL              =
000003                 =  LL        EE        VV  VV  EE       LL              =
000003                 =  LL        EEEEE      V V    EEEEE    LL              =
000003                 =  LL        EE         VVVV   EE       LL              =
000003                 =  LL        EE          VV    EE       LL              =
000003                 =  LLLLLLL   EEEEEEE     VV    EEEEEEE  LLLLLLL         =
000003                 =                                                       =
000003                 =                                                       =
000003                 =                                                       =
000003                 =                                                       =
000003                 =  PPPPPP    RRRRRR    00000    CCCCC    SSSSSS         =
000003                 =  PP   PP   RR   RR  00   00  CC   CC  SS              =
000003                 =  PP   PP   RR   RR  00   00  CC       SS              =
000003                 =  PPPPPP    RRRRRR   00   00  CC        SSSSS          =
000003                 =  PP        RRRR     00   00  CC           SS          =
000003                 =  PP        RR RR    00   00  CC   CC      SS          =
000003                 =  PP        RR  RR    00000    CCCCC   SSSSSS          =
000003                 =                                                       =
000003                 =                                                       =
000003                 =                                                       =
000003                 --------------------------------------------------------*)
000003
000003
000003
000003
000003
000003  (*=====================================================*)
000003     PROCEDURE PUTCH (
000003                       VAR OUTFILE : TEXT;    (* WHERE TO SEND THE CHARACTER *)
000003                       IT : INTEGER   (* THE INTEGER REPRESENTATION TO BE CONVERTED *)
000004                      );
000004
000004     (* USER DESCRIPTION:
000004        YOU SHOULD USE THIS PROCEDURE IF YOU WANT TO CONVERT
000004        THE INTEGER REPRESENTATION OF AN ASCII CHARACTER TO
000004        THE DISPLAY CODE REPRESENTATION OF THAT CHARACTER.
000004        IF ONE IS IN ASCII MODE (I.E. BY ISSUING THE ASCII
000004        COMMAND TO TELEX), THEN THE ASCII CHARACTER WILL
000004        BE DISPLAYED ON THE TERMINAL. THIS INCLUDES ALL CONTROL
000004        CODES. IN ADDITION THE NUMBER 128 CAUSES A WRITELN.
000004     *)
000004
000004     (* INTERNAL DESCRIPTION:
000004        THIS PROCEDURE USES A COMBINATION OF TABLE LOOKUP AND CONDITIONALS.
000004     *)
000004
000004  BEGIN
000004           IF (IT >= 0) AND (IT < 32) THEN WRITE(OUTFILE,'^',DSSPTR^.CHARS[IT+96])
000032           ELSE IF IT = 58 THEN WRITE(OUTFILE,'@' ,DSSPTR^.CHARS[68])
000054           ELSE IF IT = 64 THEN WRITE(OUTFILE,'@' ,DSSPTR^.CHARS[65])
000076           ELSE IF IT = 94 THEN WRITE(OUTFILE,'@',DSSPTR^.CHARS[66])
000120           ELSE IF (IT > 31)AND(IT < 96) THEN WRITE(OUTFILE,DSSPTR^.CHARS[IT])
000142           ELSE IF IT = 96 THEN WRITE(OUTFILE,'@',DSSPTR^.CHARS[71])
000164           ELSE IF (IT > 96) AND (IT < 123) THEN
000170              WRITE(OUTFILE,'^',DSSPTR^.CHARS[IT-32])
000213           ELSE IF (IT > 122) AND (IT < 128) THEN
000217              WRITE(OUTFILE,'^',DSSPTR^.CHARS[IT-75])
000243           (*ELSE IF IT = 128 THEN WRITELN(OUTFILE)*)
000243        END;                                      (* ** PUTCH ** *)
000247
000247
000247  (*===============================================*)
000247     FUNCTION CNVT (
000002                 VAR IOFILE : TEXT;  (* INPUT FILE BEING CONVERTED *)
000003                 VAR CH : CHAR    (* CHARACTER TO BE CONVERTED *)
000004               ) : INTEGER;
000005
000005     (* USER DESCRIPTION:
000005        THIS FUNCTION IS USED TO CONVERT A CHARACTER TO AN ITEGER
000005        IS THE ASCII ORD OF THE CHARACTER. SINCE THE DISPLAY
000005        CODE REPRESENTATION OF SOME ASCII CHARACTERS ARE 2 CHARACTER
000005        SEQUENCES, THIS FUNCTION NEEDS TO KNOW ABOUT THE TEXT
000005        FILE FROM WHICH THE CHARACTER WAS OBTAINED. NOTE THAT
000005        IF THIS FUNCTION DOES READ IN ANOTHER CHARACTER, THE
000005        PARAMETER CH IS CHANGED TO IT.
000005     *)
000005
000005
000005     (* INTERNAL DESCRIPTION:
000005        THE PROCESS OF CONVERSION IS STRIGHTFORWARD. IF THE CURRENT
000005        CHARACTER IS EITHER A ^ OR AN @, THEN THE DISPLAY CODE
000005        REPRESENTATION IS A 2 CHARACTER SEQUENCE. WE NEED TO
000005        READ IN THE SECOND CHARACTER FROM THE FILE IOFILE.
000005     *)
000005
000005  BEGIN
000005        IF CH='^' THEN BEGIN
000006           READ(IOFILE,CH);
000016           IF (ORD(CH)>=32) THEN CNVT := ORD(CH) - 32
000021                            ELSE CNVT := ORD(CH)+96
000023           END
000024        ELSE IF CH='@' THEN BEGIN
000027           READ(IOFILE,CH);
000037           CASE CH OF
000041              'A' : CNVT := 64;
000043              'B' : CNVT := 94;
000045              'D' : CNVT := 58;
```

```
000047              'G' : CNVT := 96
000047              END
000061           END
000061        ELSE IF CH = ':' THEN CNVT := 58
000064        ELSE IF CH IN ['0'..'9'] THEN CNVT := ORD(CH) + 21
000070        ELSE IF CH IN ['A'..'Z'] THEN CNVT := ORD(CH) + 64
000075                              ELSE CNVT := DSSPTR^.CONV[CH]
000106        END;                          (* CNVT *)
000115
000115
000115 (**********************************************)
000115    FUNCTION CNVTA (
000002                    BUF : INBUF;    (* ARRAY OF CHARACTERS BEING CONVERTED *)
000003                    VAR IND : INTEGER  (* INDEX INTO THE ARRAY BUF *)
000004                  ) : INTEGER;
000023
000023    (* USER DESCRIPTION:
000023       THIS FUNCTION IS LIKE THE FUNCTION CNVT, BUT IS USED WHEN
000023       THE CHARACTERS TO BE CONVERTED ARE IN AN ARRAY. THIS TIME,
000023       IF THERE IS A NEED TO GO TO ANOTHER CHARACTER, THE
000023       INDEX INTO THE ARRAY IND IS INCREMENTED.
000023    *)
000023
000023    (* INTERNAL DESCRIPTION:
000023       SEE THE DESCRIPTION FOR CNVT.
000023    *)
000023
000023
000023    VAR
000023        CH : CHAR;
000024
000024 BEGIN
000024        CH := BUF[IND];
000024        IF CH='^' THEN BEGIN
000026           IND := IND + 1;
000027           CH := BUF[IND];
000043           IF (ORD(CH)>=32) THEN CNVTA := ORD(CH) - 32
000044                            ELSE CNVTA := ORD(CH)+96
000046           END
000047        ELSE IF CH='@' THEN BEGIN
000052           IND := IND + 1;
000054           CH := BUF[IND];
000067           CASE CH OF
000070              'A' : CNVTA := 64;
000072              'B' : CNVTA := 94;
000074              'D' : CNVTA := 58;
000076              'G' : CNVTA := 96
000076              END
000110           END
000110        ELSE IF CH = ':' THEN CNVTA := 58
000113        ELSE IF CH IN ['0'..'9'] THEN CNVTA := ORD(CH) + 21
000117        ELSE IF CH IN ['A'..'Z'] THEN CNVTA := ORD(CH) + 64
000123                              ELSE CNVTA := DSSPTR^.CONV[CH]
000133        END;                          (* CNVTA *)
000143
000143
000143 (**********************************************)
```

```
000143      FUNCTION DSPCHARCNT (
000002                        VAR ARR: DYNAMIC PASBUF;  (* ARRAY THAT HAS THE CHARCTERS TO BE COUNTED *)
000003                        LEN : INTEGER     (* LENGTH OF STRING TO BE COUNTED *)
000004                       ) : INTEGER;
000011
000011      (* USER DESCRIPTION :
000011         THIS FUNCTION COUNTS THE NUMBER OF ASCII CHARACTERS IN
000011         THE ARRAY ARR. CDC CYBER DISPLAY CODE IS A 6 BIT CODE
000011         SO, IN ORDER TO REPRESENT THE FULL ASCII CHARACTER
000011         SET, TWO DISPLAY CODE CHARCTERS ARE USED FOR THE LOWER
000011         CASE CHARCTERS & THE CONTROL CHARCTERS. THE CHARACTERS ^
000011         AND AT ARE USED AS ESCAPE CHARACTERS
000011      *)
000011
000011      (* INTERNAL DESCRIPTION:
000011         OBVIOUS. LOOP TROUGH INDEXING INTO ARRAY. IF IT IS ONE
000011         OF THE MAGIC CHASRACTERS @  OR ^ THEN THIS IS A TWO CHARACTER
000011         DISPLAY CODE REPRESENTATION FOR THE ASCII CHARACTER.
000011      *)
000011
000011
000011      VAR
000011         DCNT : INTEGER;
000012         I : INTEGER;
000013
000013      BEGIN
000013         DCNT := 0;
000013         I := 1;
000014         WHILE I <= LEN DO BEGIN
000017             IF (ARR[I] = '^') OR (ARR[I] = '@') THEN BEGIN
000046                 I := I + 2;
000050             END
000050             ELSE BEGIN
000051                 I := I + 1;
000053             END;
000053             DCNT := DCNT + 1;
000055         END;
000056         DSPCHARCNT := DCNT;
000060      END;
000074
000074
000074    (*==============================================*)
000074      FUNCTION CNTCHARS (
000002                       LABL : LABELS;  (* THE ARRAY THAT CONTAINS THE TARGET LABEL *)
000003                       LABELLEN : INTEGER  (* THE LENGTH OF THE LABEL IN DISPLAY CODE CHARACTERS *)
000004                     ) : INTEGER;
000014
000014      (* USER INTERFACE:
000014         THE PURPOSE OF THIS FUNCTION IS TO TELL CREATETARG THE
000014         DIMENSIONS OF THE LABEL FOR A TOUCH TARGET. THIS NOW
000014         ONLY COMPUTES THE X LENGTH, BUT IN THE FUTURE THIS FUNCTION
000014         SHOULD BECOME A PROCEDURE AND COMPUTE THE Y LENGTH AS
000014         WELL.
000014      *)
000014
000014      (* INTERNAL DESCRIPTION:
000014         CALLS DSPCHARCNT FOR NOW, BUT IN THE FUTURE, IF TARGETS
000014         GET THE ABILITY TO HAVE MULTI-LINES AS WELL AS GRAPHICS,
000014         THIS FUNCTION WILL BECOME MUCH MORE INVOLVED.
000014      *)
000014
000014      VAR
000014          I : INTEGER;
000015
000015      BEGIN
000015          CNTCHARS := DSPCHARCNT(LABL,LABELLEN);
000014      END;
000025
000025
000025  (*==============================================*)
000025      FUNCTION DISABLED (
000002                          ID : INTEGER;    (* BLOCK ID TO CHECK *)
000003                          VAR I : INTEGER      (* THE INDEX WHERE ID WAS FOUND IN THE DISARM ARRAY *)
000004                        ): BOOLEAN;    (* TRUE IF ID WAS DISABLED *)
000005
000005      (* USER DESCRIPTION:
000005         THIS FUNCTION CHECKS TO SEE IF THE BLOCK, ID, IS DISABLED.
000005         IF IT IS, IT SETS I TO THE INDEX INTO THE DISARM ARRAY.
000005      *)
000005
000005      (* INTERNAL DESCRIPTION:
000005         LOOP THROUGH DISARM ARRAY UNTIL YOU FIND ID. RETURN THE
000005         INDEX INTO THE ARRRY FOR ID AS I.
000005      *)
000005
000005
000005      VAR
000005          HIT : BOOLEAN;
000006
000006      BEGIN
000006          I := 1;
000005          HIT := DSSPTR^.DISARM[I] = ID;
000021          WHILE (I < BLOCKMAX) AND (NOT HIT) DO BEGIN
000025              I := I + 1;
000027              HIT := DSSPTR^.DISARM[I] = ID;
000041          END;
000042          DISABLED := HIT;
000045      END;
000055
000055
000055  (*==============================================*)
000055      FUNCTION NEST (
000002                       XS,YS,XE,YE : INTEGER ;   (* THE START AND END POINTS OF THE BLOCK
000006                                                    TO CHECK FOR NESTING IN THE FOLLOWING BLOCK *)
000006                       XS1,YS1,XE1,YE1 : INTEGER      (* THE NESTING BLOCK *)
000012                   ) : NESTTYPE;      (* NESTED, REVNESTED, OR OVERLAP *)
000013
000013      (* USER DESCRIPTION:
000013         THIS FUNCTION CHECKS FOR THE NESTING OF THE BLOCK DESCRBED
000013         BY THE FIRST SET OF ENDPOINTS IN THE BLOCK DESCRIBED
000013         IN THE SECOND SET OF ENDPOINTS. THE FUNCTION RETURNS THE
000013         SCALER TYPE NESTYPE INDICATING WHETHER THE BLOCKS ARE
000013         NESTED, REVERSE NESTED, OR OVERLAPPING.
000013      *)
```

```
000013
000013      (* INTERNAL DESCRIPTION:
000013        THE LOGIC IS SELF DESCRIPTIVE.
000013      *)
000013
000013
000013      BEGIN
000013          IF ((XS1 <= XS) AND
000006              (XE1 >= XE) AND
000007              (YS1 <= YS) AND
000011              (YE1 >= YE)      ) THEN BEGIN
000014              NEST := NESTED;
000015          END
000015          ELSE BEGIN
000016              IF ((XS <= XS1) AND
000020                  (XE >= XE1) AND
000021                  (YS <= YS1) AND
000023                  (YE >= YE1)     ) THEN BEGIN
000025                  NEST := REVNESTED;
000026              END
000026              ELSE BEGIN
000027                  IF ((XE1 <= XS) OR
000031                      (XE <= XS1) OR
000032                      (YE1 <= YS) OR
000034                      (YE <= YS1)    ) THEN BEGIN
000036                      NEST := NOTNESTED;
000037                  END
000037                  ELSE BEGIN
000040                      NEST := OVERLAP;
000042                  END;
000042              END;
000042          END;
000042      END;
000065
000065
000065   (*********************************************************)
000065      PROCEDURE CONVMETRIC (
000002                             X,Y : INTEGER;    (* THE VALUES TO BE CONVERTED *)
000004                             M : METRIC;  (* THE METRIC THESE VALUES ARE IN:
000005                                          DOTS, CHS, OR PARTS *)
000005                             VAR XS,YS : INTEGER  (* THE CONVERTED VALUES IN DOTS *)
000007                            );
000007
000007      (* USER DESCRIPTION:
000007         THIS UTILITY PROCEDURE CONVERTS ONE TYPE OF SCALE FOR
000007         SPECIFYING THE DIMENSIONS OF LINES ETC. ENTERED BY A
000007         USER TI THE STANIARD FORM OF IN TERMS CF DOTS. THE
000007         USER CAN HAVE THE SCALE IN DOTS, CHARACTERS, OR FRACTIONS
000007         OF A SCREEN.
000007      *)
000007
000007
000007      (* INTERNAL DESCRIPTION:
000007         JUST A CASE STATEMENT. THE WIERD NUMBERS FOR FRACTIONS
000007         OF A SCREEN IS FOR INSURING THAT THE NUMBER 512 IS NEVER
000007         GENERATED. THUS 0% IS 0 AND 100% IS 511 NOT 512. ALSO
000007         FRACTIONS OF A SCREEN IS ENTERED IN TERMS OF PERCENT.
000007        *)
000007
000007        BEGIN
000007           CASE M OF
000005              DOTS : BEGIN
000005                 XS := X;
000006                 YS := Y;
000010              END;
000011
000011              CHS : BEGIN
000011                 XS := X * 8;
000013                 YS := Y * 16;
000015              END;
000016
000016              PARTS : BEGIN
000016                 XS := (5120 * X) DIV 1001;
000026                 YS := (5120 * Y) DIV 1001;
000036              END;
000037           END;                         (* OF CASE *)
000044        END;                            (* OF CONVMETRIC *)
000060
000060
000060      (*=============================================*)
000060        PROCEDURE CENTERIT (
000002                           X,XLEN,Y,YLEN : INTEGER;     (* X AND Y ARE THE START DISTANCE
000006                                                           AWAY FROM THE ORIGIN SPECIFIED IN THE NEXT PARM. *)
000006                          CENTERING : ADJ;      (* ORIGIN: LL,UL,LR,UR,CENTER *)
000007                          VAR XS,YS : INTEGER  (* THE END POINTS COMPUTED WITH THE LENGTHS XLEN,YLEN *)
000011                          );
000011
000011        (* USER DESCRIPTION:
000011           THIS PROCEDURE IS USED TO COMPUTE THE END POINTS FROM
000011           THE START POINTS AND LENGTHS GIVEN WITH RESPECT TO AN ORIGIN SPECIFIED.
000011           THE ORIGIN CAN BE ONE OF THE FOUR CORNERS OF A BLOCK
000011           OR ITS CENTER.
000011        *)
000011
000011        (* INTERNAL DESCRIPTION:
000011           THESE ARE TRIVIAL CALCULATIONS.
000011        *)
000011
000011
000011        BEGIN
000011           CASE CENTERING OF
000006              CENTER : BEGIN
000006                 XS := X - (XLEN DIV 2);
000011                 YS := Y - (YLEN DIV 2);
000013              END;
000014
000014              LL : BEGIN
000014                 XS := X;
000016                 YS := Y;
000017              END;
000020
000020              UL : BEGIN
000020                 XS := X;
000022                 YS := Y - YLEN;
```

```
000023              END;
000024
000024              LR : BEGIN
000024                  XS := X - XLEN;
000026                  YS := Y;
000027              END;
000030
000030              UR : BEGIN
000030                  XS := X - XLEN;
000032                  YS := Y - YLEN;
000033              END;
000034          END;                          (* OF CASE *)
000042      END;                              (* OF CENTERIT *)
000062
000062
000062   (*=============================================*)
000062      PROCEDURE MODE (
000002                    S1 : MAJORSTATE;    (* DSPTEXT OR GRAPHICS *)
000003                    S2 : MINORSTATE;    (* DSPNONE, TOUCH, PLACE, POINT,
000004                                         LINE, CHPLOT, CHLOAD *)
000004                     WMODE : INTEGER;   (* 3 ( = CONST ERASE )
000005                                           4 ( = CONST DSPNORM )
000005                                           6 ( = CONST OVER )
000005                                           16 ( = CONST INVERSE ) *)
000005                     PAGE : INTEGER     (* 0 ( = CONST DSPNO )
000006                                         1 ( = CONST DSPYES ) *)
000006                    );
000006
000006      (* USER DESCRIPTION :
000006         THIS PROCEDURE GET THE ORION TERMINAL INTO ITS PROPER
000006         STATE FOR DOING VARIOUS ACTIONS. THE MAJOR DIVISION
000006         IS BETWEEN GRAPHICS AND TEXT. THE MINOR DIVISION IS
000006         FOR VARIOUS GRAPHIC ACTIONS AS WELL AS LOADING CHARACTERS
000006         INTO THE ORION. WHENEVER THIS PROCEDURE IS CALLED,
000006         (AND IT SHOULD BE CALLED WHEN YOU WISH TO DO SOMETHING
000006         DIFFERENT FROM WAHT YOU WERE DOING BEFORE), YOU HAVE THE
000006         OPTION OF SETTING THE WRITING MODE (SOME ACTIONS, LIKE
000006         LINE DRAWING AND POINT PLOTTING, HAVE ONLY TWO POSSIBILE
000006         WRITING MODES (NORMAL OR ERASE), OTHERS HAVE NONE
000006         (CHLOAD IN WHICH CASE THE WRITING MODE CAN BE SET
000006         TO ANYTHING). THERE IS ALSO THE OPTION OF CLEARING THE
000006         SCREEN BEFORE SETTING THE MODE AND THIS IS CONTROLLED
000006         BY PAGE. THIS IS PROVIDED BECAUSE IT WAS PROVIDED BY
000006         THE ORION PROTOCALL AND I WANTED TO PRESERVE AS MUCH
000006         CAPABILITY AS POSSIBLE AT THIS LEVEL. FINALLY, WHEN
000006         SETTING THE MAJOR STATE TO TEXT, SET THE MINOR STATE
000006         TO NONE.
000006      *)
000006
000006      (* INTERNAL DESCRIPTION:
000006         THIS PROCEDURE WORKS BY HAVING THE PRIOR STATE OF THE
000006         TERMINAL REMEMBERED IN TWO GLOBAL VARIABLES. IT THEN
000006         COMPARES THE CURRENT STATE WITH THE PRIOR STATE. IF
000006         THERE IS A DIFFERENCE IN THE MAJOR STATE, THEN THE
000006         SEQUENCE TO GET THE ORION INTO TEXT OR GRAPHICS IS
000006         SENT. IF THERE IS A DIFFERENCE IN THE MINOR STATE,
000006         (OR YOU WANT TO CLEAR THE SCREEN OR CHANGE THE WRITING
000006        MODE WHILE IN GRAPHICS) THEN THE LOAD MODE 3 CHARACTER
000006        SEQUENCE IS SENT. THIS SEQUENCE LOOKS LIKE:
000006
000006           CHARACTER 1 : /1/001/0/XX/  (X IS DON'T CARE )
000006           CHARACTER 2 : /1/000000/
000006           CHARACTER 3 : /1/X/MM/PP/S/ WHERE
000006                                                 MM = 0 - POINT PLOT
000006                                                      1 - VECTOR PLOT
000006                                                      2 - LOAD MEMORY
000006                                                      3 - CHARACTER PLOT
000006
000006                                                 PP = 0 - ERASE
000006                                                      1 - WRITE
000006                                                      2 - INVERSE
000006                                                      3 - OVERSTRIKE
000006
000006                                                  S = 0 - NOERASE
000006                                                      1 - ERASE
000006
000006        FINALLY, IF THE MODE YOU ARE IN IS TEXT, AND YOU SET
000006        THE WRITE MODE DIFFERENTLY OR YOU SET PAGE, THEN
000006        THESE ACTIONS ARE SIMULATED TO INCREASE THE ORTHAGONALITY
000006        OF THE SEMANTICS.
000006     *)
000006
000006
000006         PROCEDURE SETUP( SECMODE : INTEGER ; WMODE,PAGE : INTEGER);
000005         VAR
000005             W : INTEGER;
000006             FUNC : INTEGER;
000007
000007         BEGIN
000007             CASE WMODE OF
000005                 3: W := 0;
000007                 4: W := 1;
000011                 6: W := 3;
000013                 16: W := 2;
000015             END;
000030             PUTCH(OUTPUT,LF);
000032             PUTCH(OUTPUT,64);    (* AN @  *)
000034             FUNC := 64 + W*2 + PAGE + SECMODE;
000040             PUTCH(OUTPUT,FUNC);
000042         END;
000056
000056
000056
000056     BEGIN
000056         IF S1 < DSSPTR^.SAVE1 THEN BEGIN     (*GO INTO TEXT MODE*)
000012
000012             (* THIS SEQUENCE IS FROM THE WORK OF PELLETT *)
000012             PUTCH(OUTPUT,07);
000014             PUTCH(OUTPUT,127);
000016             PUTCH(OUTPUT,127);
000020             PUTCH(OUTPUT,DSPCR);
000022             PUTCH(OUTPUT,BEL);
000024             PUTCH(OUTPUT,BEL);
000026             PUTCH(OUTPUT,BEL);
```

```
000030         END;
000030         IF S1 > DSSPTR^.SAVE1 THEN BEGIN     (*GO INTO GRAPHICS *)
000037             PUTCH(OUTPUT,GS);
000041         END;
000041         IF S1 <> DSPTEXT THEN BEGIN
000043             IF (S2 <> DSSPTR^.SAVE2) OR (PAGE = DSPYES) OR (DSSPTR^.SAVEMODE <> WMODE) THEN BEGIN
000061                 CASE S2 OF
000062                     POINT: BEGIN
000062                         SETUP(0,WMODE,PAGE);
000064                         DSSPTR^.SAVEMODE := WMODE;
000072                     END;
000073                     LINE : BEGIN
000073                         SETUP(8,WMODE,PAGE);
000075                         DSSPTR^.SAVEMODE := WMODE;
000103                     END;
000104                     CHPLOT : BEGIN
000104                         SETUP(24,WMODE,PAGE);
000106                         DSSPTR^.SAVEMODE := WMODE;
000114                     END;
000115                     CHLOAD : BEGIN
000115                         SETUP(16,WMODE,PAGE);
000117                     END;
000120                     OTHERWISE
000126                 END;            (* OF CASE *)
000126             END;
000126         END
000126         ELSE BEGIN
000127             IF PAGE = DSPYES THEN PUTCH(OUTPUT,CLEAR);
000133             IF (DSSPTR^.SAVEMODE <> WMODE) OR (PAGE = DSPYES) THEN PUTCH(OUTPUT,WMODE);
000145             DSSPTR^.SAVEMODE := WMODE;
000153         END;
000153         DSSPTR^.SAVE1 := S1;
000162         DSSPTR^.SAVE2 := S2;
000171     END;                        (* OF MODE*)
000203
000203  (*=================================================*)
000203     PROCEDURE SETCOORD (
000002                        X3,Y3 : INTEGER;   (* X AND Y COORDINATES *)
000004                        M : METRIC     (* DOTS,CHS,OR PARTS *)
000005                      );
000005
000005     (* USER DESCRIPTION:
000005        NOTHING SPECIAL ABOUT THIS ONE.
000005     *)
000005
000005     (* INTERNAL DESCRIPTION:
000005        SEND OUT THE SET COORIDINATE SEQUENCE AS FOLLOWS:
000005
000005            CHARACTER 1 : /0/010/XXX/
000005            CHARACTER 2 : /1/XX/C/A8A7A6/
000005            CHARACTER 3 : /1/A5A4A3A2A1A0/
000005                                           WHERE
000005                                                 C = 0 - SET X COORD
000005                                                     1 - SET Y COORD
000005                                                 AN(N=7..0) ARE ADDRESS BITS
000005
000005        THIS SEQUENCE IS SENT OUT TWICE, ONCE FOR X, ONCE FOR Y.
000005       *)
000005
000005       VAR
000005          X,Y : INTEGER;
000007          X1 , X2 , Y1 , Y2 : INTEGER;
000013       BEGIN
000013          CONVMETRIC(X3,Y3,M,X,Y);
000010          MODE(GRAPHICS,PLACE,DSPNORM,DSPNO);
000015          PUTCH(OUTPUT,DLE);
000017          X1 := (X DIV 64) + 64;
000023          PUTCH(OUTPUT,X1);
000025          X2 := (X MOD 64) + 64;
000031          PUTCH(OUTPUT,X2);
000033          PUTCH(OUTPUT,DLE);
000035          Y1 := (Y DIV 64) + 72;
000041          PUTCH(OUTPUT,Y1);
000043          Y2 := (Y MOD 64) + 64;
000047          PUTCH(OUTPUT,Y2);
000051       END;
000075
000075     (*********************************************************)
000075       PROCEDURE DRAWLINE (
000002                             X3,Y3 : INTEGER;    (* END POINT COORDS *)
000004                             M : METRIC;    (* DOTS,CMS, OR PARTS *)
000005                             WMODE : INTEGER;    (* THE WRITING MODE, SEE MODE *)
000006                             PAGE : INTEGER      (* CLEAR SCREEN FLAG, SEE MODE *)
000007                           );
000007
000007       (* USER DESCRIPTION:
000007          DRAW A LINE FROM THE LAST POINT THE CURSOR WAS SET TO
000007          THE ENDPOINTS X3,Y3. YOU CAN SET THE CURSOR BY ISSUING
000007          A SETCOORD BEFORE CALLING THIS PROCEDURE. BEWARE! IF
000007          YOU WANT TO ERASE A LINE, DRAW THE LINE EXACTLY THE
000007          SAME WAY YOU DID IT ORIGINALLY (BUT NOW IN ERASE MODE)
000007          THIS IS BECAUSE THE LINE DRAWING ROUTINE IN ROM IN THE
000007          ORION WILL DRAW LINES DIFFERENTLY IF THE START AND END
000007          POINTS ARE INTERCHANGED. FINALLY, THE ONLY VALID WRITING
000007          MODES FOR LINES ARE ALLSNORM AND ERASE.
000007       *)
000007
000007       (* INTERNAL DESCRIPTION:
000007          SEND OUT THE 3 CHARACTER ADDRESS SEQUENCE AS FOLLOWS:
000007
000007             CHARACTER 1: /1/X8X7X6X5X4X3/
000007             CHARACTER 2: /1/X2X1X0Y8Y7Y6/
000007             CHARACTER 3: /1/Y5Y4Y3Y2Y1Y0/
000007
000007          THIS SEQUENCE IS THE SAME FOR DRAWPOINT.
000007       *)
000007
000007
000007       VAR
000007          X,Y : INTEGER;
000011          X1 : INTEGER;
000012          X2ANDY1 : INTEGER;
000013          Y2 : INTEGER;
000014
```

```
000014      BEGIN
000014          CONVMETRIC(X3,Y3,M,X,Y);
000010          MODE(GRAPHICS,LINE,WMODE,PAGE);
000016          X1 := (X DIV 8) + 64;
000021          PUTCH(OUTPUT,X1);
000023          X2ANDY1 := 64 + (X MOD 8)*8 + (Y DIV 64);
000030          PUTCH(OUTPUT,X2ANDY1);
000032          Y2 := (Y MOD 64) + 64;
000036          PUTCH(OUTPUT,Y2);
000040      END;
000066
000066   (*==============================================*)
000066      PROCEDURE DRAWPOINT (
000002                            X3,Y3 : INTEGER;   (* WHERE THE POINT GOES *)
000004                            M : METRIC;    (* DOTS, CHS, OR PARTS *)
000005                            WMODE : INTEGER;   (* WRITE MODE *)
000006                            PAGE : INTEGER     (* SCREEN PAGE FLAG *)
000007                           );
000007
000007      (* USER DESCRIPTION:
000007         DRAW A POINT AT X3,Y3. ONLY DSPNORM AND ERASE MAKE
000007         SENSE FOR WMODE.
000007      *)
000007
000007      (* INTERNAL DESCRIPTION:
000007         SEE DRAWLINE.
000007      *)
000007
000007
000007      VAR
000007          X,Y : INTEGER;
000011          X1 : INTEGER;
000012          X2ANDY1 : INTEGER;
000013          Y2 : INTEGER;
000014
000014      BEGIN
000014          CONVMETRIC(X3,Y3,M,X,Y);
000010          MODE(GRAPHICS,POINT,WMODE,PAGE);
000016          X1 := (X DIV 8) + 64;
000021          PUTCH(OUTPUT,X1);
000023          X2ANDY1 := 64 + (X MOD 8)*8 + (Y DIV 64);
000030          PUTCH(OUTPUT,X2ANDY1);
000032          Y2 := (Y MOD 64) + 64;
000036          PUTCH(OUTPUT,Y2);
000040      END;
000066
000066
000066   (*==============================================*)
000066      PROCEDURE PUTCHAR (
000002                           CH : INTEGER;   (* ORD OF CHARACTER TO DRAW *)
000003                           CHMODE : MEMORY;    (* NORMAL,PROGRAMMABLE *)
000004                           WMODE : INTEGER;    (* WRITE MODE, SEE MODE *)
000005                           PAGE : INTEGER  (* CLEAR SCREEN FLAG, SEE MODE *)
000006                          );
000006
000006      (* USER DESCRIPTION:
000006         THIS PROCEDURE "DRAWS" A CHARACTER ATE THE CURRENT CURSOR
000006         POSITION. YOU PASS THE INTEGER ORD OF THE CHARACTER
000006         TO BE DRAWN INSTEAD OF THE CHARACTER ITSELF SINCE THIS
000006         PROCEDURE CAN ALSO DRAW IN THE ALTERNATE CHARACTER SET
000006         AS SPECIFIED BY THE PARAMETER MEMORY. I THOUGHT IT WOULD
000006         BE MUCH MORE USEFULL TO MAP ALTERNATE CHARACTERS TO
000006         INTEGERS AS OPPOSED TO MAPPING THEM TO THE STANDARD
000006         ASCII CHARACTER SET. YOU SHOULD USE THIS ROUTINE TO
000006         DISPLAY THE ALTERNATE CHARACTERS WHOSE ORDS ARE < 32
000006         SINCE THESE ARE NOT ACCESSABLE IN TEXT MODE. (IF YOU
000006         ATTEMPT TO DO THIS, THEY WILL BE INTERPRETED AS BEING
000006         ASCII CONTROL CHARACTERS BY THE ORION.) NOTE: FOR NORMAL
000006         PRINTING OF TEXT, YOU DON'T HAVE TO USE THIS ROUTINE.
000006        IN FACT, THIS ROUTINE IS SLOWER (BY A FACTOR OF 3)
000006        THAN DOING A SIMPLE WRITE.
000006      *)
000006
000006      (* INTERNAL DESCRIPTION:
000006         THIS PROCEDURE SPITS OUT 3 CHARACTER CHARACTER PLOT
000006         SEQUENCES PER CHARACTER PLOTTED. THE FIRST CHARACTER
000006         IS A "CONTROL ACCESS" ESCAPE CHARACTER TELLING THE
000006         ORION THAT THE FOLLOWING CHARACTER IS FOR CONTROL.
000006         THE SECOND CHARACTER IS A CONTROL CHARACTER THAT
000006         SELECTS THE CHARACTER FONT MEMORY. THERE ARE FOUR
000006         POSSIBILITIES HERE: LOW ROM, HIGH ROM, LOW RAM, AND
000006         HIGH RAM. THE LAST CHARACTER IS THE CHARACTER TO BE
000006         PLOTTED WITH ITS MSB STRIPPED. (THE MSB IS INCORPORATED
000006         IN THE MEMORY SELECT CHARACTER). KLUDGY, HEH? I DID
000006         NOT IMPLIMENT ALL THE FREATURES OF THE "CONTROL ACCESS"
000006         ESCAPE SEQUENCE. THE ORION ALLOWS FOR MOST OF THE ASCII
000006         CONTROL CHARACTERS THAT HAVE MEANING TO THE ORION TO
000006         BE ALSO CARRIED OUT BY THIS ESCAPE SEQUENCE. SINCE
000006         ONE IS IN GRAPHICS MODE, ONE DOES NOT HAVE TO GET OUT
000006         OF GRAPHICS TO MOVE DOWN A LINE, SAY. BUT, $%&, WHAT
000006         A PAIN IN THE &*= !!! ALSO, IF ONE REMEMBERS WHAT
000006         MEMORY WAS LAST SELECTED, THEN ONE COUID AVOID SENDING
000006         THE ESCAPE SEQUENCE. THIS WOULD ALLOW FOR 3 CHARACTERS
000006         TO BE DRAWN, BUT COULD YOU IMAGINE THE TROUBLE IF I
000006         TRIED TO MAINTAIN THIS CAPABILITY AT THIS LEVEL OF
000006         SOFTWARE! ENOUGH BITCHING, THE 3 CHARACTER SEQUENCE
000006         IS AS FOLLOWS:
000006
000006             CHARACTER 1 : /1/C5C4C3C2C1C0/ I USE THIS TO SEND OUT A CONTROL ACCESS
000006                                            CHARACTER, HEX 7F.
000006             CHARACTER 2 : /1/C5C4C3C2C1C0/ I USE THIS TO SEND OUT A SELECT MEMORY
000006                                            CHARACTER, WHERE:
000006                                            C5..0 = 16 - LOW ROM
000006                                                    17 - HIGH ROM
000006                                                    18 - LOW RAM
000006                                                    19 - HIGH RAM
000006             CHARACTER 3 : /1/C5C4C3C2C1C0/ THE CHARACTER WITH MSB STRIPPED.
000006
000006      *)
000006
000006
000006      VAR
000006          ST : INTEGER;
000007
```

```
000007      BEGIN
000007         IF CHMODE = NORMAL THEN ST := 16 ELSE ST := 18;
000011         MODE(GRAPHICS,CHPLOT,WMODE,PAGE);
000017         IF CH >= 64 THEN BEGIN
000022             PUTCH(OUTPUT,DELETE);
000024             PUTCH(OUTPUT,ST + 64 + 1);
000030             PUTCH(OUTPUT,CH);
000032         END
000032         ELSE BEGIN
000033             PUTCH(OUTPUT,DELETE);
000035             PUTCH(OUTPUT,ST + 64);
000040             PUTCH(OUTPUT,CH + 64);
000043         END;
000043      END;
000057
000057
000057  (*********************************************)
000057      PROCEDURE DRAWCHAR (
000002                          CH : INTEGER;      (* INTEGER ORD OF CHARACTER TO BE DRAWN *)
000003                          X1,Y1 : INTEGER;   (* WHERE TO DRAW CHARACTER *)
000005                          M : METRIC;    (* DOTS, CHS, OR PARTS *)
000006                          CENTERING : ADJ;   (* LL,LR,UL,UF, OR CENTER *)
000007                          CHMODE : MEMORY;   (* NORMAL, PROGRAMMABLE *)
000010                          WMODE  : INTEGER;      (* WRITE MODE, SEE MODE *)
000011                          PAGE : INTEGER     (* CLEAR SCREEN FLAG, SEE MODE *)
000012                        );
000012
000012      (* USER DESCRIPTION:
000012         DRAW A CHARACTER WHOSE ORD IS CH AT POINT X,Y. THE
000012         REFERENCE POINT IS SPECIFIED BY CENTERING. FOR NORMAL
000012         CHARACTER PLOTTING, SUGGEST USING CENTER FOR CENTERING.
000012         YOU  CAN ALSO PLOT PROGRAMMABLE CHARACTERS AS SPECIFIED
000012         BY MEMORY. THIS ROUTINE IS MENT TO BE A BETTER USER
000012         INTERFACE THAN PUTCHAR. SEE COMMENTS OF PUTCHAR.
000012      *)
000012
000012      (* INTERNAL DESCRIPTION:
000012         THIS ROUTINE SIMPLY CALLS SETCOORD BEFORE CALLING
000012         PUTCHAR.
000012      *)
000012
000012
000012      VAR
000012         X,Y : INTEGER;
000014         XS, YS : INTEGER;
000016
000016      BEGIN
000016         CONVMETRIC(X1,Y1,M,X,Y);
000011         CENTERIT(X,8,Y,16,CENTERING,XS,YS);
000021         SETCOORD(XS,YS,DOTS);
000026         PUTCHAR(CH, CHMODE, WMODE, PAGE);
000033      END;
000065
000065
000065  (*********************************************)
000065      PROCEDURE LOADCHRS (
000002                         CHS : ROM;      (* DATA MATRIX OF CHARACTERS *)
000003                      START : INTEGER;    (* START ADDRESS IN ORION TO START PLACING CHARACTERS *)
000004                      NUM : INTEGER  (* NUMBER OF CHARACTERS *)
000005                    );
000605
000605        (* USER DESCRIPTION:
000605           PROGRAM THE PROGRAMMBALE MEMORY IN THE ORION. THE CHARACTERS
000605           ARE IN THE MATRIX CHS. THIS MATRIX IS [# OF CHARS, 8]
000605           THE VALUES IN THIS MATRIX REPERSENT ONE VERTICAL SLICE
000605           OF A CHARACTER. CHARACTERS ARE IN A 16X8 FORMAT. THUS,
000605           THE LARGEST VALUE THAT SHOULD BE IN CHS IS (2**17)-1.
000605           THE PROCEDURE GETSET IS USED TO LOAD MATRICIES OF THIS TYPE
000605           FROM AN EXTERNAL FILE. YOU CAN SPECIFY WHERE THE PROGRAMMABLE
000605           CHARACTER SET IS TO START IN RAM OF THE ORION BY START.
000605           YOU MAY WANT TO DO THIS TO BIAS THE START OF THE PROGRAMMABLE
000605           CHARACTER SET TO ABOVE THE CONTROL CHARACTERS.
000605        *)
000605
000605        (* INTERNAL DESCRIPTION:
000605           SPIT OUT THE LOAD ADDRESS 3 CHARACTER SEQUENCE, THEN
000605           PROCEED TO SEND OUT LOAD CHARACTER SEQUENCES UNTIL ALL
000605           CHARACTERS ARE TRANSFERRED. NOTE THAT YOU LOAD 1 COLUMN
000605           AT A TIME. TO THE ORION, EVERYTHING IS IN TERMS OF THESE
000605           VERTICAL SLICES, INCLUDING THE START ADDRESS. I CONVERT
000605           FOR YOU THE START ADDRESS IN TERMS OF CHARACTERS TO ORION'S TERMS.
000605           THE 3 CHARACTER LOAD ADDRESS SEQUENCE IS AS FOLLOWS:
000605
000605               CHARACTER 1 : /0/100/XXX/
000605               CHARACTER 2 : /1/XX/A9A8A7A6/
000605               CHARACTER 3 : /1/A5A4A3A2A1A0/ WHERE A9..0 IS THE 10 BIT COLUMN ADDRESS
000605                                              TO START LOADING TO.
000605
000605           THE 3 CHARACTER COLUMN LOAD SEQUENCE IS AS FOLLOWS:
000605
000605               CHARACTER 1 : /1/XX/B16B15B14B13/
000605               CHARACTER 2 : /1/B12B11B10B9B8B7/
000605               CHARACTER 3 : /1/B6B5B4B3B2B1/   WHERE B16..1 ARE THE 16 BITS OF A COLUMN.
000605
000605        *)
000605
000605
000605        VAR
000605            ST, ST1, ST2 : INTEGER;
000610            C1, C2, C3 : INTEGER;
000613             T : INTEGER;
000614             I,J : INTEGER;
000616
000616        BEGIN
000616            MODE(GRAPHICS,CHLOAD,ERASE,DSPNO);
000016            (*LOAD ADDRESS*)
000016            PUTCH(OUTPUT,32);        (* ASCII SPACE *)
000020            ST := START * 8;
000023            ST1 := (ST DIV 64) + 64;
000025            PUTCH(OUTPUT,ST1);
000027            ST2 := (ST MOD 64) + 64;
000033            PUTCH(OUTPUT,ST2);
000035
000035             (*TRANSFER ROM*)
```

```
000035          FOR I := 0 TO NUM DO BEGIN
000042              FOR J := 0 TO 7 DO BEGIN
000046                  T := CHS[I,J];
000061                  C1 := (T DIV 4096) + 64;
000063                  PUTCH(OUTPUT,C1);
000065                  C2 := ((T DIV 64) MOD 64) + 64;
000072                  PUTCH(OUTPUT,C2);
000074                  C3 := (T MOD 64) + 64;
000100                  PUTCH(OUTPUT,C3);
000102              END
000102          END;
000113      END;
000143
000143
000143 (*============================================*)
000143      PROCEDURE GETLN (
000002                        VAR INFILE : TEXT;      (* INPUT FILE *)
000003                        VAR CH : INBUF;    (* ARRAY TO PLACE LINE IN *)
000004                        VAR LEN : INTEGER;     (* LENGTH OF LINE *)
000005                        VAR DONE : BOOLEAN     (* HIT EOF FLAG *)
000006                       );
000006
000006      (* USER DESCRIPTION:
000006         THIS UTILITY PROCEDURE GETS A LINE FORM THE TEXT FILE
000006         INFILE. IT PLACES THE TEXT IN THE ARRAY INBUF AND SETS
000006         LEN TO THE LENGTH OF THE FILE. NOTE THAT IF THE LINE IS LARGER
000006         THAN THE BUFFER SIZE, THE REST OF THE LINE CAN BE
000006         STILL RETRIEVED BY CALLING GETLN AGAIN. FOR THIS
000006         REASON, YOU ARE RESPONSIBLE FOR ISSUING A READLN
000006         AFTER CALLING GETLN. DONE IS SET TRUE IF THE EOF IS
000006         REACHED.
000006      *)
000006
000006      (* INTERNAL DESCRIPTION:
000006         NOTHING SPECIAL GOING ON HERE. THIS ROUTINE USES THE
000006         GLOBAL CONSTANT, BUFLEN, TO TELL WHEN TO QUIT STUFFING
000006         THE BUFFER.
000006      *)
000006
000006
000006      BEGIN
000006          LEN := 1;
000005          DONE := EOF(INFILE);
000011          IF NOT DONE THEN BEGIN
000013             IF NOT EOLN(INFILE) THEN BEGIN
000014                 READ(INFILE,CH[LEN]);
000035                 WHILE(NOT (EOLN(INFILE) OR (LEN = BUFLEN))) DO BEGIN
000042                     LEN := LEN +1;
000043                     READ( INFILE, CH[LEN]);
000064                 END;
000065             END
000065             ELSE BEGIN
000066                 LEN := 0;
000067             END;
000067          END;
000067      END;
000075
```

```
000075  (*=================================================*)
000075     PROCEDURE GETSET (
000002                          VAR CHTEXT : TEXT;     (* WHICH FILE THE CHARACTER SET IS FROM *)
000003                          VAR CHARSET : ROM;     (* THE MATRIX TO PLACE THE CHARACTER SET *)
000004                          VAR CHNO : INTEGER     (* THE NUMER CF CHARACTERS READ IN *)
000005                        );
000005
000005
000005     (* USER DESCRIPTION:
000005     THIS PROCEDURE FETCHES THE CHARACTER SET LYING CN THE
000005     TEXT FILE CHTEXT ANC PLACES IT IN THE MATRIX CHARSET. CHARSET IS
000005     A 128X8 ARRAY. EACH COLUMN OF THIS ARRAY REPRESENTS ONE OF THE 8
000005     COLUMNS OF A CHARACTER. EACH ARRAY ELEMENT IS A ROW OF
000005     A CHARACTER. THE PARAMETER CHNO TELLS YOU HOW MANY CHARACTERS
000005     WAS LOADED INTO CHARSET. THE STANDARD ALTERNATE CHARACTER
000005     SET IS ILLUSTARTED BELOW. INCREASING INDICIES OF THE
000005     CHARSET MATRIX IS ACCROSS THE PAGE:
000005      *)
```

```
000005       (*$L'ALTERNATE SET'*)
000005       (*
000005
000005
000005                                  00     00      0C         00           0000              0       00     00      000     000
000005                                 0          0       0       0            0                 0      0  0     0         0        0
000005       000000                    0          0       C     0              0                 0      0  0     0        0       0
000005            0   00000     00     0          0       C     0       00     0     00    0   00000    0  0     0       0          0
000005            0  00  0     0  0   0            0      0     0       00     0     0     0     0       00     000     C000   000
000005                0  0    0    0   0          0        0    0              0     0     0   00000
000005                0  0    0    0   0          0       C     0     000000   0     0     0     0
000005                0  0     0  0    0          0       0     0              0     00    0     0
000005                0  0      00      00     00         0     0       00     0     0 000       0
000005                                                    0    0        00     0     0
000005                                                 00    00                0     0
000005                                                                         0000
000005
000005
000005
000005
000005         0 0     0000    0       0000    0000     00       0    00  00 000000                                000000
000005         0 0     0       0          0    0  0    0  0      0 0   00  00 0   0                       00  0    0  000000
000005         0000    000     000       0      00      0CC    C   0          0   0   00000      0       0     0      0    0    00 00
000005           0        0    0  0     0      0  0       0                   0   0              0       0      0      0  0   0  0  0
000005           0     000      00      0      00C0       0                   0   0      0     00000      0       0     0  0   0  0  0
000005                                                                        0   0      0       0       0        0      00     00 00
000005                                                                        0   0   00000      0       0       0       00
000005                                                                        0   0      0               0      0
000005                                                                       000 000     0     00000     0     0
000005                                                                                                  00      0    0
000005                                                                                                          000000
000005
000005
000005
000005
000005
000005       0 000 0   000                      00                               0                 0     0       00     000000    000
000005       0  0  0    0                0      0C      0    0        0          0     00          0       0     0  0    0   0   0   0
000005       0  0  0   000              0      0  0    00C    0      C 0 0       0    0  0       0         0     0  0    0      0     0
000005        0 0 0   0 0 0    0       0       0  0    0 C 0    0       0        0         0     0         0     0  0    0      0     0
000005         000    0 0 0   0       C        0  0      0      0                0         0     0         0     0 0     0      0 000 0
000005          0     0 0 0  000000    0      0     0    0      0    C0000    0 0 0   00  0     0         0      0      0      0     0
000005          0      000    0       0 0     0     0    0    0 0              000    0 00      0         0     0 0     0      0     0
000005          0       0      0       0 0    0     0    C    0 0               0     0   0     0         0   0   0     0       0   0
000005         000     000              0    000  000    0    0      00000            0  0      0         0   0    0  000         000
000005                                   0                   0                        C0        0         0
000005                                                                                           0       0
000005                                                                                            0     0
000005
000005
000005       *)
```

```
000005  (*$L'ALTERNATE SET'*)
000005  (*
000005
000005
000005  000000           000000       00             000    00 00                                                       000       0
000005  0     0          0     0     0  0     0     0   0  0  0   0         0 0             0            0 0           0    0     0
000005   0           0               0      0 C    0     0 0   0  0          00           0              0 0           0    0   0 0 0
000005    0           0   0   0     000      0 0   0     0     0     0        0      0000000     00      0 0    000  0 0 00     0 0 0
000005      0     000000  0000       0      0   0  0     0     0    0 0 0     00        0       0000     0 0   0   00  0    0   0 0 0
000005     0          0   0   0      0      0   0   0   0      0       0      0 0    0000000    0000     0 0   0   0   0     0   000
000005   0           0             0 0    0      0   0 0       0              00       0          00     0 0   0  0 0  0     0    0
000005  0     0          0     0  0 0   0 0      0 0 0 0 0     0    00000     0      0                   0 0    00   0 00   0     0
000005  000000           000000     0 000  0000000  000 000   000             00                                        0 00       0
000005                                                                      0 0                                        0
000005
000005
000005
000005
000005
000005
000005      0                      00                                                  00                              00     0   0
000005      0                       0                                                     0                               0   0  0 0
000005      0             00         0    0 000             00000       00                0                                0 0  0    0
000005    0 0    000     0 0         0    00    0    00              0 0      0 0 0       0             00       0000        0   0    0
000005   0 0 0   0   0      0   0     0   0     0     0     00000   0 0     0  0   0      0  00     0   0  0    0    0      0    00000
000005   0 0 0   0           00       0 0  0     0     0             00     0  0   0       0   0    0 0     0  0    0      0    0    0
000005    0 0     00                0  0  0     0     0     00000   0 0     0  0   0      0     0   0 0     0  0    0      0    0    0
000005      0    0                 0    0 0     0     0             0  0    0  0   0      0      0  0 0     0  00  0      0      0 0
000005      0    0   0            0     0       0     00            0   0    00 00        0       00    0  0    0 00       0       0
000005      0     000                           0                                         0             00      0.
000005                                                                                 00                       0
000005                                                                                                          0
000005
000005
000005
000005
000005                    0000                                                         0000
000005                   0                                                                 0     0 0
000005      00           0                     0   0    0            00000   000           0     00    00 00      0
000005   000    000000    00       0    0     0 0   0  0   00    0    0     00000          0     0    0 0         0
000005  0    0  0 0      0          0 0    000       00     0    0   0      00000          0    00    0 0         0
000005  0    0     0     0           0    0    0     00     0    0   0      00000          0   0 0     00 00      0
000005  0    0     0     0          0 0   0    0    0  0    0    0   0       000           0    00                0
000005   000       0     .0000     0    0 0    0   0    0   0   0    0  00                 0     0                0
000005             00          0           000           0   00       00  0                0     00             00000
000005                      00                                           0                 0     0 0
000005                                                                 00                  0
000005                                                                                 0000
000005
000005     *)
```

```
000005  (*$L'ALTERNATE SET'*)
000005  (*
000005                                                   0   0    0   0   0   0  00  00  00  00  00  00 000 000 000 000 000 000
000005   0   0   0   0   0   0   00  00  00  00 000 C00 000 000 000 000 000 000 000 000 000 000 00000000000000000000000000000000
000005            0   0  00  00  00  00  000 0C0 00C C00 0CC 000 000 000 000 000 000 0000000000000000000000000C00000000000000000
000005                                                           0   0   00  00  00  00  00  00  00  00  00  00  000 0000000000
000005                                                   0   0    0   0   0   0  00  00  00  00  00  00 000 000 000 000 000 000
000005   0   0   0   0   0   0   00  00  00  00 0C0 C00 000 000 000 000 000 000 000 000 000 000 00000000000000C00000000000000000
000005            0   0  00  00  00  00  000 000 000 000 000 C00 000 000 000 000 C00 0000000000000000000000000000000000000000000
000005                                                           0   0   00  00  00  00  00  00  00  00  00  00  000 0000000000
000005                                                   0   0    0   0   0   0  00  00  00  00  00  00 000 000 000 000 000 000
000005   0   0   0   0   0   0   00  00  00  00 000 CC0 000 C00 C00 000 000 000 000 000 000 000 000000000000000000000000000000C0
000005            0   0  00  0C  00  C0  000 000 000 000 000 000 000 000 000 000 000 0000000000000000000000000000000000000000000
000005                                                           0   0   00  00  00  00  00  00  00  00  00  00  000 0000000000
000005                                                   0   0    0   0   0   0  00  00  00  00  00  00 000 000 000 000 000 000
000005   0   0   0  0    0   0   00  00  00  00 000 000 000 000 C00 000 000 000 0C0 000 000 000 00000000000000000000000000000000
000005            0  0   00  00  00  00  000 000 000 C00 0C0 000 000 000 000 000 000 0000000000000000000000000C000C0000000000000
000005                                                           0   0   00  00  C0  00  00  00  00  00  00  00  000 0000000000
000005  00000000
000005  00000000
000005  000000000            0
000005  0000000000          00
000005  00000000000        000
000005  00000000 000      000
000005  00000000  000    000
000005  00000000   000  000
000005  00000000  000    000
000005  00000000 000      000
000005  00000000000        000
000005  0000000000          00
000005  000000000            0
000005  00000000
000005  00000000
000005  00000000
000005  *)
000005
```

```
000005  (*$L'LOW LEVEL ROUTINES'*)
000005
000005
000005     (* INTERNAL DESCRIPTION:
000005        DO A RESET ON THE TEXT FILE CHTEXT, THEN READ IN CHARACTERS
000005        IN THIS FORM FROM THE TEXT FILE:
000005
000005
000005
000005
000005
000005
000005  OOOOOO
000005       O
000005       O
000005  127 NOT
000005
000005
000005
000005    OOOOO
000005  OO  O
000005   O  O
000005   O  O
000005   O  O
000005   O  O
000005  128 PI
000005
000005
000005
000005     OO
000005    O  O
000005   O    O
000005   O    O
000005    O  O
000005     OO
000005  129 CIRCLE
000005
000005
000005        THE LINES WITH THE NUMBERS AND IDENTIFIERS ON THEM ARE
000005        IGNORED. I GOT THIS CHARACTER SET FROM THE DCS UNIX
000005        SYSTEM AND IT WAS USED FOR THE GOULD PLOTTER. IN THE
000005        FUTURE THESE NUMBERS COULD BE USED AS INDECIES INTO THE
000005        ARRAY CHSET, BUT THIS WAS A QUICK AND DIRTY IMPLEMENTATION.
000005        NOTE THAT THE ROWS OF THE CHARACTER ARE STORED IN CHSET
000005        AS BINARY BITS TO A 16 BIT WORD.
000005     *)
000005     VAR
000005          I, J, K, L, M, MM : INTEGER;
000013          DONE : BOOLEAN;
000014          LFN : INTEGER;
000015          CH : INBUF;
000033
000033     BEGIN
000033         CHNO := -1;
000006         K := -1;
000007         FOR I := 0 TO 127 DO BEGIN
000013             FOR J := 0 TO 7 DO BEGIN
```

```
000017                 CHARSET[I,J] := 0;
000033             END;
000037         END;
000043         RESET(CHTEXT);
000045         GETLN(CHTEXT,CH,LEN,DONE);
000051         READLN(CHTEXT);
000054         WHILE NOT DONE DO BEGIN
000056             IF (CH[1] = ' ') OR (CH[1] = 'O') OR (LEN = 0) THEN BEGIN
000065                 K := K + 1;
000067                 IF LEN > 0 THEN BEGIN
000071                     IF LEN > 8 THEN LEN := 8;
000073                     FOR J := 1 TO LEN DO BEGIN
000107                         DSSPTR^.BUILD[15-K,J-1] := CH[J];
000125                     END;
000131                 END;
000131             END
000131             ELSE BEGIN
000132                 IF K > 0 THEN BEGIN
000134                     CHNO := CHNO + 1;
000136                     FOR L := 0 TO 7 DO BEGIN
000142                         MM := 1;
000144                         FOR M := 0 TO 15 DO BEGIN
000147                             IF DSSPTR^.BUILD[M,L] = 'O' THEN CHARSET[CHNO,L] := CHARSET[CHNO,L] + MM;
000210                             MM := MM * 2;
000212                             DSSPTR^.BUILD[M,L] := ' ';
000223                         END;
000227                     END;
000233                 END;
000233                 K := -1;
000235             END;
000235             GETLN(CHTEXT,CH,LEN,DONE);
000241             IF NOT DONE THEN READLN(CHTEXT);
000246         END
000246     END;
000273
000273
000273 (*==============================================*)
000273     PROCEDURE DRAWBOX (
000002                        X3,XL,Y3,YL : INTEGER;   (* THE ORIGIN   LENGTH OF THE BOX *)
000006                        M1,M2 : METRIC;  (* SCALE FOR ORIGIN   LENGTHS *)
000010                        CENTERING : ADJ;     (* LL,UL,LR,UR & CENTER *)
000011                        THICKNESS : INTEGER;     (* THICKNESS OF BORDER
000012                                                  - GROWS OUTWARD
000012                                                  + GROWS INWARD
000012                                                  > 512 FILL AREA IN COMPLETELY *)
000012                        WMODE : INTEGER;      (* WRITE MODE, SEE MODE *)
000013                        PAGE : INTEGER    (* CLEAR SCREEN FLAG, SEE MODE *)
000014                      );
000014
000014      (* USER DESCRIPTION :
000014         THIS PROCEDURE DRAWS RECTANGLERS THAT HAVE AN ORIGIN AS SPECIFIED
000014         BY X3,Y3 AND THE PARAMETER CENTERING. CENTERING TELLS
000014         YOU WHERE THE POINT X3,Y3 IS RELATIVE TO THE RECTANGLE. THE
000014         LENGTHS OF THE RECTANGLE IS SPECIFIED BY XL,YL. THE
000014         THICKNESS OF THE DRAWN BOARDERS OF THE RECTANGLE IS SPECIFIED
000014         BY THICKNESS. IF THICKNESS IS A POSITIVE NUMBER, THEN
000014         THE BOARDER IS GROWN INWARDS FROM THE EDGE OF THE RECTANGLE.
```

```
000014          IF THICKNESS IS NEGATIVE, THEN THE BOARDER IS GROWN
000014          OUTWARDS FROM THE EDGE OF THE RECTANGLE. IF THICKNESS
000014          IS > 512 THEN THE RECTANGLE IS FILLED COMPLETELY, I.E.
000014          YOU WANT A SOLID FIGURE.
000014      *)
000014
000014      (* INTERNAL DESCRIPTION:
000014         NOTHING SPECIAL ABOUT THIS ROUTINE.
000014      *)
000014
000014
000014      VAR
000014          X,XLEN,Y,YLEN : INTEGER;
000020          XS,YS,X1,X2,Y1,Y2,I,INC,T : INTEGER;
000031
000031      BEGIN
000031          CONVMETRIC(X3,Y3,M1,X,Y);
000011          CONVMETRIC(XL,YL,M2,XLEN,YLEN);
000017          CENTERIT(X,XLEN,Y,YLEN,CENTERING,XS,YS);
000030          IF PAGE = DSPYES THEN DRAWPOINT(XS,YS,DOTS,WMODE,DSPYES);
000037          IF THICKNESS > 512 THEN BEGIN    (* FILL IN RECTANGLE COMPLETLY*)
000042              FOR I := 0 TO (XLEN -1) DO BEGIN
000046                  SETCOORD(XS + I, YS,DOTS);
000053                  DRAWLINE(XS + I, YS + YLEN -1,DOTS, WMODE, DSPNO);
000062              END;
000067          END                          (*OF FULL RECTANGLE*)
000067          ELSE BEGIN
000070              IF THICKNESS > 0 THEN INC := 1 ELSE INC := -1;
000075              T := ABS(THICKNESS);
000100              X1 := XS;
000102              Y1 := YS;
000103              X2 := XS + (XLEN - 1);
000105              Y2 := YS + (YLEN - 1);
000110              FOR I := 1 TO T DO BEGIN
000114                  SETCOORD(X1,Y1,DOTS);
000121                  DRAWLINE(X2,Y1,DOTS,WMODE,DSPNO);
000127                  DRAWLINE(X2,Y2,DOTS,WMODE,DSPNO);
000135                  DRAWLINE(X1,Y2,DOTS,WMODE,DSPNO);
000143                  DRAWLINE(X1,Y1,DOTS,WMODE,DSPNO);
000151                  X1 := X1 + INC;
000153                  Y1 := Y1 + INC;
000155                  X2 := X2 - INC;
000156                  Y2 := Y2 - INC;
000160              END
000160          END;                         (*OF PARTIAL FILL CASE*)
000164      END;
000244
000244
000244      PROCEDURE CREATETARG (
000002                              BLOCKID : INTEGER;    (* BLOCK FOR TARGET *)
000003                              ID : INTEGER;    (* USER SPECIFIED TARGET ID *)
000004                              X2,XL2 : INTEGER;     (* X ORIGIN AN LENGTH OF TARGET *)
000006                              Y2,YL2 : INTEGER;     (* Y ORIGIN AND LENGTH *)
000010                              M1,M2 : METRIC;  (* METRICS FOR THE ABOVE DIMENSIONS *)
000012                              CENTERING : ADJ;      (* LL,UL,LR,UR,CENTER *)
000013                              AUTOMAKE : BOOLEAN;  (* TRUE IF TARGET SIZE IS TO BE MADE TO FIT THE LABEL *)
000014                              AUTOINCREMENT :BOOLEAN;  (* TRUE IF THE PLACEMENT OF THE TARGET IS TO BE MADE AUTOMATIC *)
```

```
000015                                 STYLE : TARGTYPE;     (* FAT,UNDERLINE *)
000016                                 LABL : LABELS;    (* THE ARRAY WITH THE TARGET LABEL *)
000017                                 LBLEN : INTEGER   (* THE LENGTH OF THE LABEL *)
000020                               ) ;
000027
000027               VAR
000027                   PTR,PTR1 : TPTR;
000031                   I,J : INTEGER;
000033                   X,XLEN,Y,YLEN,XS,YS : INTEGER;
000041                   XBOX,YBOX,XCENTER,YCENTER : INTEGER;
000045                   XBOX1,YBOX1,X1,Y1,XL,YL : INTEGER;
000053                   OUTLINE : INTEGER;
000054
000054               BEGIN
000054                   PTR1 := DSSPTR^.BLK[BLOCKID].TARGS^[1];
000025                   I := 1;
000027                   WHILE (I < TARGMAX) AND (PTR1 <> NIL) DO BEGIN
000034                       I := I + 1;
000036                       PTR1 := DSSPTR^.BLK[BLOCKID].TARGS^[I];
000055                   END;
000056                   IF PTR1 = NIL THEN BEGIN
000060                       NEW(PTR);
000062                       DSSPTR^.BLK[BLOCKID].TARGS^[I] := PTR;
000102
000102
000102                       CONVMETRIC(X2,Y2,M1,X,Y);
000110                       CONVMETRIC(XL2,YL2,M2,XLEN,YLEN);
000116                (*     CENTERIT(X,XLEN,Y,YLEN,CENTERING,XS,YS); *)
000116
000116                  (* FIND OUT THE DIMENSIONS OF THE RECTANGLE *)
000116                       IF AUTOINCREMENT THEN BEGIN
000120                           XCENTER := DSSPTR^.BLK[BLOCKID].TARGPX + (16 + 7);
000133                           YCENTER := DSSPTR^.BLK[BLOCKID].TEXPY - (16 + 7);
000144                       END
000144                       ELSE BEGIN
000145                           XCENTER := X;
000147                           YCENTER := Y;
000151                       END;
000151                       XBOX := (XCENTER + 16) DIV 32;
000155                       YBOX := (YCENTER - 16) DIV 32;
000157
000157                       (* COMPUTE SIZE IN X DIRECTION *)
000157                       XL := CNTCHARS(LABL,LBLEN) ;
000162                       XBOX1 := (XL DIV 4) + 1;     (* # OF BOXES *)
000165
000165                       (* COMPUTE SIZE IN Y DIRECTION *)
000165                       YL := 1;
000166                       YBOX1 := (YL DIV 2) + 1;
000170
000170                       (* CHECK IF WE NEED TO GO TO A NEW LINE *)
000170                       IF AUTOINCREMENT THEN BEGIN
000171                           IF DSSPTR^.BLK[BLOCKID].OUTLINE < 0 THEN OUTLINE := 0 ELSE OUTLINE := - DSSPTR^.BLK[BLOCKID].OUTLINE;
000215                           IF ((XBOX + XBOX1)*32 + (7 - 1)) >= (DSSPTR^.BLK[BLOCKID].XF + OUTLINE) THEN BEGIN
000232                               DSSPTR^.BLK[BLOCKID].TEXPY := YBOX*32 - (32 - 7);
000244                               YBOX := YBOX - 2;
000246                               DSSPTR^.BLK[BLOCKID].TARGPX := DSSPTR^.BLK[BLOCKID].TEXPX;
000265                               XBOX := (DSSPTR^.BLK[BLOCKID].TARGPX + (32 + 7)) DIV 32;
```

```
000277                    END;
000277               END;
000277               PTR^.LORGX := (((32 * XBOX1) - (8 * XL)) DIV 2) + XBOX * 32;
000311               PTR^.LORGY := (((32 * YBOX1) - (16 * YL)) DIV 2) + YBOX * 32 ;
000321
000321               (* DRAW A BOX AROUND AREA *)
000321               DRAWBOX(XBOX * 32,XBOX1 * 32,YBOX * 32,YBOX1 * 32,DOTS,DOTS,LL,-2,DSPNORM,DSPNO);
000335
000335               (* DISPLAY THE LABEL *)
000335               SETCOORD(PTR^.LORGX,PTR^.LORGY,DOTS);
000350               MODE(DSPTEXT,DSPNONE,DSPNORM,DSPNO);
000356               FOR I := 1 TO LBLEN DO BEGIN
000363                    WRITE(OUTPUT,LABL[I]);
000401               END;
000406               SETCOORD(10,10,DOTS);
000411
000411               (* RESET TEXT POINTERS *)
000411               IF AUTOINCREMENT THEN BEGIN
000413                    DSSPTR^.BLK[BLOCKID].TARGPX := (XBOX + XBOX1)*32 + ((32 -1) - 7);
000427               END;
000427
000427               (* FILL IN DATA STRUCTURES *)
000427               PTR^.TOUCHED := FALSE;
000434               PTR^.ID := ID;
000441               PTR^.STYLE := STYLE;
000452               PTR^.LABL := LABL;
000457               PTR^.LBLEN := LBLEN;
000465               PTR^.X := XBOX * 32;
000472               PTR^.XLEN := XBOX1 * 32;
000500               PTR^.Y := YBOX * 32;
000505               PTR^.YLEN := YBOX1 * 32;
000512               FOR I := 1 TO XBOX1 DO BEGIN
000516                    FOR J := 1 TO YBOX1 DO BEGIN
000523                         DSSPTR^.TARGARRAY[XBOX + I - 1,YBOX - (J - 1) ].TARG := PTR;
000537                    END;
000544               END;
000551          END
000551          ELSE BEGIN                    (* CASE OF NO ROOM FOR TARGS *)
000552          END;
000552     END;                               (* OF CREATETARG *)
000660
000660
000660 (*===============================================*)
000660     PROCEDURE DSTTARG (
000002                            BLOCKID : INTEGER;    (* BLOCK IN WHICH TARGET LIES *)
000003                            TARGID : INTEGER      (* ID OF TARGET TO DESTROY *)
000004                           );
000004
000004     (* USER DESCRIPTION:
000004        USE THIS PROCEDURE TO DESTROY TARGETS THAT HAVE BEEN
000004        CREATED USING CREATETARG. SCRAPS DATA STRUCTURE FOR
000004        TARGET AND REMOVES TARGET FROM THE SCREEN.
000004     *)
000004
000004     VAR
000004          PTR : TPTR;
```

```
000007
000007      BEGIN
000007         I := 1;
000006         WHILE (DSSPTR^.BLK[BLOCKID].TARGS^[I]^.ID <> TARGID) DO BEGIN
000032            I := I + 1;
000033         END;
000034         PTR := DSSPTR^.BLK[BLOCKID].TARGS^[I];
000053
000053         (* UNDO LABEL & BOX *)
000053         SETCOORD(PTR^.LORGX,PTR^.LORGY,DOTS);
000066         MODE(DSPTEXT,DSPNONE,DSPNORM,DSPNO);
000074         FOR I := 1 TO CNTCHARS(PTR^.LABL,PTR^.LBLEN) DO BEGIN
000114            WRITE(OUTPUT,' ');
000121         END;
000126         DRAWBOX ( PTR^.X,PTR^.XLEN,
000137                   PTR^.Y,PTR^.YLEN,
000151                   DOTS,DOTS,LL,-5,
000161                   ERASE,DSPNO
000162                  );
000164
000164         (* SCRAP DATA STRUCTURE *)
000164         FOR I := 0 TO 15 DO BEGIN
000170            FOR J := 0 TO 15 DO BEGIN
000174               IF ((DSSPTR^.TARGARRAY[I,J].TARG <> NIL ) AND
000206                  (DSSPTR^.TARGARRAY[I,J].BLOCKID = BLOCKID)) THEN BEGIN
000220                  IF (DSSPTR^.TARGARRAY[I,J].TARG^.ID = TARGID) THEN BEGIN
000234                     DISPOSE(DSSPTR^.TARGARRAY[I,J].TARG);
000245                     DSSPTR^.TARGARRAY[I,J].TARG := NIL;
000257                  END;
000257               END;
000257            END;
000264         END;
000270      END;
000304
000304
000304    (**************************************************)
000304      FUNCTION FETCH (
000002                   VAR PTR : TEXTPTR;      (* 2TUPLE POINTER TO CURRENT TEXT *)
000003                   VAR ENDT : BOOLEAN      (* INDICATES NO MORE TEXT IN BUFFERS *)
000004                 ) : CRNG;
000005
000005      (* THIS FUNCTION RETRIEVES A CHARACTER FROM THE TEXT BUFFERS
000015        IT RESETS PTR AS ONE EXTRACTS FROM THE BUFFERS *)
000005
000005      BEGIN
000005         ENDT := FALSE;
000005         PTR.POS := PTR.POS + 1;
000011         IF PTR.POS > PTR.BUF^.EPOS THEN BEGIN   (* GO TO NEXT BUFFER *)
000016            PTR.BUF := PTR.BUF^.NEXT;
000023            IF PTR.BUF <> NIL THEN BEGIN    (* CHECK FOR END OF TEXT *)
000025               PTR.POS := PTR.BUF^.POS + 1;
000034               FETCH := PTR.BUF^.AR[PTR.POS];
000053            END
000053            ELSE BEGIN
000054               ENDT := TRUE;
000055               FETCH := 0;      (* ASCII NULL *)
000057            END;
```

```
000057         END
000057         ELSE BEGIN
000060             FETCH := PTR.BUF^.AR[PTR.POS];
000077         END;
000077     END;                            (* OF FETCH *)
000104
000104  (*************************************************)
000104     PROCEDURE DSPLINE (
000002                             ID : INTEGER;   (* BLOCK ID *)
000003                             PTR : TEXTPTR;  (* POINTER TO A BUFFER TO DISPLAY *)
000004                             VAR PTR1 : TEXTPTR;     (* PTR AFTER DISPLAYED LINE *)
000005                             DMODE : LINEMODE ;  (* ERASE OR PLACE *)
000006                             PX,PY : INTEGER     (* POSITION TO START DISPLAY *)
000010                           );
000012
000012
000012     VAR
000012         X,Y,I,II : INTEGER;
000016         TTYPE : TARGTYPE;
000017         ENDT : BOOLEAN;
000020         TARGID : INTEGER;
000021         INDEX : INTEGER;
000022         LABCNT : INTEGER;
000023         CHR,CHR1 : INTEGER;
000025         ENDOFLN : BOOLEAN;
000026         WMODE : INTEGER;
000027         PX1 : INTEGER;
000030         SETAG : BOOLEAN;
000031
000031
000031
000031     BEGIN
000031         PTR1 := PTR;
000013         SETAG := FALSE;
000014
000014         IF DMODE <> CONSUME THEN BEGIN  (* CASES THAT REQUIRE SEMANTIC ACTIONS *)
000016             SETCOORD(PX,PY,DOTS);    (* PLACE CURSOR AT RIGHT PLACE *)
000022             CHR1 := FETCH(PTR1,ENDT);
000026             IF DMODE = UNDO THEN BEGIN
000030                 WMODE := ERASE;
000032             END
000032             ELSE BEGIN
000033                 WMODE := DSSPTR^.BLK[ID].WMODE;
000045             END;
000045             ENDOFLN := FALSE;
000047             PX1 := PX;
000051             WHILE ((CHR1 <> EL) AND
000053                    (CHR1 <> LI) AND
000056                    (NOT ENDOFLN ) ) DO BEGIN
000061                 IF CHR1 < STSPEC THEN BEGIN       (* JUST DISPLAY IT *)
000063                     IF PX1 <= DSSPTR^.BLK[ID].TEXTENDX THEN BEGIN   (* OK, ON SAME LINE *)
000075                         IF SETAG THEN SETCOORD(PX1,PY,DOTS);    (* GO BACK TO WHERE WE WERE IF NEED BE *)
000102                         SETAG := FALSE;
000104                         IF DMODE <> UNDO THEN BEGIN
000106                             MODE(DSPTEXT,DSPNONE,WMODE,DSPNO);
000114                             PUTCH(OUTPUT,CHR1);
000116                         END
```

```
000116                ELSE BEGIN
000117                    MODE(DSPTEXT,DSPNONE,DSPNORM,DSPNO);
000125                    WRITE(OUTPUT,' ');
000132                END;
000132                PX1 := PX1 + 8;
000134            END
000134            ELSE BEGIN  (* END OF LINE CASES *)
000135                IF DSSPTR^.BLK[ID].OVERFLOW = WRAP THEN BEGIN
000146                    ENDOFLN := TRUE;     (* STATE THAT WE ARE AT THE END *)
000147                END
000147                ELSE BEGIN
000150
000150                    (* ADVANCE TO END W/O DISPAY *)
000150                    DSPLINE(ID,PTR,PTR1,CONSUME,PX,PY);
000157                END;
000157            END;        (* OF ECL CASES *)
000157        END
000157        ELSE BEGIN
000160
000160            CASE CHR1 OF     (* SEMANTICS *)
000161
000161                GO : BEGIN  (* GRAPHICS ORIGIN *)
000161                    X := FETCH(PTR1,ENDT);
000165                    Y := FETCH(PTR1,ENDT);
000171                    IF X = 999 THEN DSSPTR^.BLK[ID].PLOTORGX := PX1 ELSE
000205                        DSSPTR^.BLK[ID].PLOTORGX := DSSPTR^.BLK[ID].TEXTORGX + X;
000225                    IF Y = 999 THEN DSSPTR^.BLK[ID].PLOTORGY := PY ELSE
000241                        DSSPTR^.BLK[ID].PLOTORGY := DSSPTR^.BLK[ID].TEXTENDY + Y;
000261                    (* SETCOORD(DSSPTR^.BLK[ID].PLOTORGX,DSSPTR^.BLK[ID].PLOTORGY,DOTS); *)
000261                END;
000262
000262                SC : BEGIN  (* SET CURSOR FOR GRAPHICS ETC. *)
000262                    X := FETCH(PTR1,ENDT);
000266                    Y := FETCH(PTR1,ENDT);
000272                    IF X = 999 THEN X := PX1 ELSE X := DSSPTR^.BLK[ID].PLOTORGX + X;
000307                    IF Y = 999 THEN Y := PY ELSE Y := DSSPTR^.BLK[ID].PLOTORGY + Y;
000324                    SETCOORD (X,Y,DOTS);
000331                END;
000332
000332                LN : BEGIN  (* DRAW A LINE *)
000332                    SETAG := TRUE;
000334                    X := FETCH(PTR1,ENDT);
000337                    Y := FETCH(PTR1,ENDT);
000343                    DRAWLINE(DSSPTR^.BLK[ID].PLOTORGX + X,
000354                             DSSPTR^.BLK[ID].PLOTORGY + Y,
000364                             DOTS,
000367                             WMODE,
000370                             DSPNO );
000372                END;
000373
000373                PT : BEGIN  (* PUT A POINT *)
000373                    SETAG := TRUE;
000375                    X := FETCH(PTR1,ENDT);
000401                    Y := FETCH(PTR1,ENDT);
000404                    DRAWPOINT(DSSPTR^.BLK[ID].PLOTORGX + X,
000415                              DSSPTR^.BLK[ID].PLOTORGY + Y,
000425                              DOTS,
000430                          WMODE,
000431                          DSPNO );
000433            END;
000434
000434            CH : BEGIN  (* DRAW A CHARACTER *)
000434                 SETAG := TRUE;
000436                 CHR := FETCH(PTR1,ENDT);
000441                 X := FETCH(PTR1,ENDT);
000445                 Y := FETCH(PTR1,ENDT);
000451                 DRAWCHAR(CHR,
000453                          DSSPTR^.BLK[ID].PLOTORGX + X,
000464                          DSSPTR^.BLK[ID].PLOTORGY + Y,
000474                          DOTS,
000476                          CENTER,
000501                          NORMAL,
000503                          WMODE,
000505                          DSPNO );
000506            END;
000507
000507
000507            CA : BEGIN  (* DRAW A CHARACTER IN ALTERNATE SET *)
000507                 SETAG := TRUE;
000511                 CHR := FETCH(PTR1,ENDT);
000514                 X := FETCH(PTR1,ENDT);
000520                 Y := FETCH(PTR1,ENDT);
000524                 DRAWCHAR(CHR,
000526                          DSSPTR^.BLK[ID].PLOTORGX + X,
000537                          DSSPTR^.BLK[ID].PLOTORGY + Y,
000547                          DOTS,
000551                          CENTER,
000554                          PROGRAMMABLE,
000556                          WMODE,
000560                          DSPNO );
000561            END;
000562
000562            ML : BEGIN  (* SET LEFT MARGIN *)
000562                 X := FETCH(PTR1,ENDT);
000566                 DSSPTR^.BLK[ID].TEXTORGX := DSSPTR^.BLK[ID].TEXTORGX + X*8;
000606            END;
000607
000607            MR : BEGIN  (* SET RIGHT MARGIN *)
000607                 X := FETCH(PTR1,ENDT);
000613                 DSSPTR^.BLK[ID].TEXTENDX := DSSPTR^.BLK[ID].TEXTENDX - X*8;
000633            END;
000634
000634            MU : BEGIN  (* SET UPPER MARGIN *)
000634                 Y := FETCH(PTR1,ENDT);
000640                 DSSPTR^.BLK[ID].TEXTORGY := DSSPTR^.BLK[ID].TEXTORGY - Y*8;
000660            END;
000661
000661            MB : BEGIN  (* SET BOTTOM MARGIN *)
000661                 Y := FETCH(PTR1,ENDT);
000665                 DSSPTR^.BLK[ID].TEXTENDY := DSSPTR^.BLK[ID].TEXTENDY + Y*8;
000705            END;
000706
000706            CR : BEGIN  (* CARRIAGE RETURN W/O LINEFEED *)
                       DSSPTR^.BLK[ID].TEXPX := DSSPTR^.BLK[ID].TEXTORGX;
```

```
000726                      SETCOORD(DSSPTR^.BLK[ID].TEXPX,
000735                               DSSPTR^.BLK[ID].TEXPY,
000744                               DOTS );
000747                  END;
000750
000750                  AL : BEGIN  (* PRINT STUFF IN ALTERNATE SET *)
000750                      IF SETAG THEN SETCOORD(PX1,PY,DOTS);     (* GO BACK TO WHERE WE WERE IF NEED BE *)
000755                      SETAG := FALSE;
000757                      CHR := FETCH(PTR1,ENDT);
000762                      WHILE CHR <> NR DO BEGIN
000765                          IF DMODE <> UNDO THEN BEGIN
000767                              PUTCHAR(CHR,
000770                                      PROGRAMMABLE,
000771                                      DSSPTR^.BLK[ID].WMODE,
001002                                      DSPNO);
001004                          END
001004                          ELSE BEGIN
001005                              MODE(DSPTEXT,DSPNONE,DSPNORM,DSPNO);
001013                              WRITE(OUTPUT,' ');
001020                          END;
001020                          CHR := FETCH(PTR1,ENDT);
001024                      END;     (*OF WHILE *)
001025                  END;
001026
001026                  OV : BEGIN  (* OVERSTRIKE *)
001026                      IF SETAG THEN SETCOORD(PX1,PY,DOTS);     (* GO BACK TO WHERE WE WERE IF NEED BE *)
001033                      SETAG := FALSE;
001035                      CHR := FETCH(PTR1,ENDT);
001040                      IF WMODE <> ERASE THEN BEGIN
001043                          MODE(DSPTEXT,DSPNONE,OVER,DSPNO);
001050                          PUTCH(OUTPUT,BACK);      (* BACKSPACE *)
001052                          PUTCH(OUTPUT,CHR);
001054                      END;
001054                  END;
001055
001055                  OA: BEGIN   (* OVERSTRIKE USING ALTERNATE SET *)
001055                      IF SETAG THEN SETCOORD(PX1,PY,DOTS);     (* GO BACK TO WHERE WE WERE IF NEED BE *)
001062                      SETAG := FALSE;
001064                      CHR := FETCH(PTR1,ENDT);
001067                      IF WMODE <> ERASE THEN BEGIN
001072                          PUTCH(OUTPUT,BACK);      (* BACKSPACE *)
001074                          PUTCHAR(CHR,
001076                                  PROGRAMMABLE,
001100                                  OVER,
001101                                  DSPNO);
001102                      END;
001102                  END;
001103
001103                  UN : BEGIN  (* UNDERLINE TEXT *)
001103                      IF SETAG THEN SETCOORD(PX1,PY,DOTS);     (* GO BACK TO WHERE WE WERE IF NEED BE *)
001110                      SETAG := FALSE;
001112                      IF DMODE <> UNDO THEN BEGIN
001114                          MODE(DSPTEXT,DSPNONE,DSPNORM,DSPNO);
001122                          I := 0;
001124                          CHR := FETCH(PTR1,ENDT);
001127                          WHILE CHR <> UE DO BEGIN
001132                              I := I + 1;
```

```
001114                         PUTCH(OUTPUT,CHR);
001135                         CHR := FETCH(PTR1,ENDT);
001141                     END;     (* OF WHILE *)
001142                     FOR II := 1 TO I DO BEGIN    (* BACKUP *)
001147                         PUTCH(OUTPUT,BACK);
001151                     END;
001156                     FOR II := 1 TO I DO BEGIN    (* UNDERLINE *)
001162                         WRITE(OUTPUT,'_');
001167                     END;
001174                 END
001174                 ELSE BEGIN
001175                     MODE(DSPTEXT,DSPNONE,DSPNORM,DSPNO);
001203                     CHR := FETCH(PTR1,ENDT);
001207                     WHILE CHR <> UF DO BEGIN
001212                         WRITE(OUTPUT,' ');
001216                         CHR := FETCH(PTR1,ENDT);
001222                     END;
001223                 END;
001223             END;
001224
001224             ND : BEGIN  (* SET TO NORMAL DESTRUCTIVE *)
001224                 DSSPTR^.BLK[ID].WMODE := DSPNORM;
001235                 IF DMODE <> UNDO THEN WMODE := DSPNORM;
001240                 MODE(DSPTEXT,DSPNONE,DSSPTR^.BLK[ ID ].WMODE,DSPNO);
001256             END;
001257
001257             NP : BEGIN  (* SET TO NORMAL PROTECTIVE *)
001257                 DSSPTR^.BLK[ID].WMODE := OVER;
001270                 IF DMODE <> UNDO THEN WMODE := OVER;
001273                 MODE(DSPTEXT,DSPNONE,DSSPTR^.BLK[ ID ].WMODE,DSPNO);
001311             END;
001312
001312             RD : BEGIN  (* SET TO REVERSE DESTRUCTIVE (INVERSE ) *)
001312                 DSSPTR^.BLK[ID].WMODE := INVERSE;
001321                 IF DMODE <> UNDO THEN WMODE := INVERSE;
001326                 MODE(DSPTEXT,DSPNONE,DSSPTR^.BLK[ ID ].WMODE,DSPNO);
001344             END;
001345
001345             RP : BEGIN  (* SET TO REVERSE PROTECTIVE (ERASE) *)
001345                 DSSPTR^.BLK[ID].WMODE := ERASE;
001356                 WMODE := ERASE;
001357                 MODE(DSPTEXT,DSPNONE,DSSPTR^.BLK[ ID ].WMODE,DSPNO);
001374             END;
001375
001375             TT,TU : BEGIN   (* THE TWO TARGET TYPES *)
001375                 SETAG := TRUE;
001377                 IF WMODE <> ERASE THEN BEGIN
001401                     IF CHR1 = TT THEN TTYPE := FAT ELSE TTYPE := UNDERLINE;
001406                     TARGID := FETCH(PTR1,ENDT);
001412                     INDEX := FETCH(PTR1,ENDT);
001416                     LABCNT := FETCH(PTR1,ENDT);
001422                     CREATETARG(ID,TARGID,
001424                                 0,0,0,0,DOTS,DOTS,
001432                                 LI,TRUE,TRUE,
001440                                 TTYPE,
001443                                 DSSPTR^.LABARR
                                     LABCNT );
```

```
001460                           END
001460                           ELSE BEGIN
001461                               TARGID := FETCH(PTR1,ENDT);
001465                               INDEX := FETCH(PTR1,ENDT);
001471                               LABCNT := FETCH(PTR1,ENDT);
001475                               DSTTARG(ID,TARGID);
001500                           END;
001500                       END;
001501
001501                       OTHERWISE    (* NOTHING *)
001522
001522                   END;         (* OF CASE *)
001522               END;             (* OF IF *)
001522               CHR1 := FETCH(PTR1,ENDT);
001526           END;                 (* OF WHILE *)
001527       END
001527       ELSE BEGIN               (* JUST CONSUME UNTIL END OF LINE *)
001530           CHR := FETCH(PTR1,ENDT);
001534           WHILE (CHR <> EL) AND (CHR <> LI) DO BEGIN
001541               CHR := FETCH(PTR1,ENDT);
001545           END;
001546       END;
001546   END;                         (* OF DSPLINE *)
001616
001616
001616 (*==============================================*)
001616   PROCEDURE GETTOUCHINP (
000002                           VAR X,Y : INTEGER;  (* X,Y COORDINATES OF TOUCH *)
000004                           VAR CBUF : INBUF;   (* ARRAY OF INPUT CHARACTERS *)
000005                           VAR LEN : INTEGER;  (* LENGTH OF STRING IN CHARRAY *)
000006                           MKCIRCLE : BOOLEAN;      (* MAKE A CIRCLE
000007                                                    WHERE USER TOUCHED *)
000007                           VAR ERROR : ERRORTYPE    (* BADTOUCH *) );
000010
000010   FORWARD;
000010
000010
000010 (*==============================================*)
000010   PROCEDURE HANDLEEOL(ID : INTEGER     (* BLOCK ID *)); (* HANDLE NEXT LINE CASES *)
000003
000003   VAR
000003       PX,PY : INTEGER;
000005       PTR,PTR1 : TEXTPTR;
000011       TEMP : DSPBUFPTR;
000012       CHARR : INBUF;
000030       X,Y : INTEGER;
000032       LEN : INTEGER;
000033       ERROR : ERRORTYPE;
000034
000034   BEGIN
000034       DSSPTR^.BLK[ID].TEXPY := DSSPTR^.BLK[ID].TEXPY - 16;    (* GO DOWN A LINE *)
000024       IF DSSPTR^.BLK[ID].TEXPY <= DSSPTR^.BLK[ID].TEXTENDY THEN BEGIN  (* END OF PAGE CONDITION *)
000043
000043           (* PAUSE AND ALLOW USER TO READ THE SCREEN *)
000043           (* USER TOUCHES SCREEN TO CONTINUE         *)
000043           GETTOUCHINP(X,Y,CHARR,LEN,FALSE,ERROR);
000052
```

```
000052             CASE DSSPTR^.BLK[ID].OVERFLOW OF
000063
000063                 NOSCROLL: BEGIN     (* ERASE SCREEN THEN PRINT LINE *)
000063                     PX := DSSPTR^.BLK[ID].TEXTORGX;
000073                     PY := DSSPTR^.BLK[ID].TEXTORGY;
000102                     PTR.BUF := DSSPTR^.BLK[ID].HEAD;
000113                     PTR.POS := DSSPTR^.BLK[ID].HEAD^.POS;
000130                     WHILE (PTR.BUF <> DSSPTR^.BLK[ID].LSTPTR.BUF) OR
000142                            (PTR.POS <> DSSPTR^.BLK[ID].LSTPTR.POS) DO BEGIN
000153                         DSPLINE(ID,PTR,PTR1,UNDO,PX,PY);    (* UNDO LINE *)
000161                         WHILE PTR.BUF <> PTR1.BUF DO BEGIN
000164                             TEMP := PTR.BUF^.NEXT;
000171                             DISPOSE(PTR.BUF);
000173                             PTR.BUF := TEMP;
000175                             PTR.POS := PTR.BUF^.POS;
000204                         END;     (* OF WHILE *)
000205                         PTR := PTR1;
000207                         PY := PY  - 16;     (* ADVANCE LINE POINTER *)
000211                     END;
000212                     DSSPTR^.BLK[ID].HEAD := DSSPTR^.BLK[ID].LSTPTR.BUF;
000231                     DSSPTR^.BLK[ID].HEAD^.POS := DSSPTR^.BLK[ID].LSTPTR.POS;
000254                     DSSPTR^.BLK[ID].TEXPX := DSSPTR^.BLK[ID].TEXTORGX;
000272                     DSSPTR^.BLK[ID].TEXPY := DSSPTR^.BLK[ID].TEXTORGY;
000310                 END;            (* OF NOSCROLL *)
000311
000311                 SCROLL : BEGIN  (* SCROLL LINES *)
000311                 END;
000312
000312             END;                (* OF CASE *)
000316         END;                    (* OF IF *)
000316     END;                        (* OF HANDLEEOL *)
000340
000340
000340 (*============================================*)
000340 PROCEDURE INITDSPARRAYS  (
000002                            CHFILE : ALFA    (* EXTERNAL FILE NAME OF ALTERNATE CHARACTER SET *)
000003                          ):
000003
000003    (* USER DESCRIPTION:
000003       THIS PROCEDURE SHOULD BE CALLED ONCE BEFORE
000003       ANY OF THE DISPLAY ROUTINES ARE CALLED. THIS
000003       IS A DATA DEFINITION PROCEDURE.
000003    *)
000003
000003    (* INTERNAL DESCRIPTION:
000003       NOTHING SURPRISING ABOUT THIS PROCEDURE.
000003       CLEARS THE SCREEN WHEN CALLED.
000003    *)
000003
000003
000003
000003 VAR
000003       I,J,K,L             : INTEGER;
000007       ALTCHARSET : TEXT;
000043       CHMEM : MOM;
000643       CHNO : INTEGER;
```

```
000644         NEW(DSSPTR);              (* CREATE THE LOCAL STATIC RECORD STRUCTURE *)
000013         DSSPTR^.SAVEMODE := DSPNORM;
000021         DSSPTR^.SAVE1 := DSPTEXT;
000025         DSSPTR^.SAVE2 := DSPNONE;
000032         FOR I := 1 TO BLOCKMAX DO DSSPTR^.DISARM[I] := 0;
000047        LINELIMIT(OUTPUT,MAXINT);
000051         DSSPTR^.BLOCKO := BLOCKMAX + 1;
000057         FOR I := 0 TO 15 DO BEGIN
000062              FOR J := 0 TO 15 DO BEGIN
000066                   DSSPTR^.TARGARRAY[I,J].TARG := NIL;
000077              END;
000103         END;
000107        FOR I := 1 TO BLOCKMAX DO DSSPTR^.BLK[I].INUSE := FALSE;
000125        FOR I := 0 TO 1 DO BEGIN
000130              FOR J := 0 TO 1 DO BEGIN
000133                    FOR K := 0 TO 1 DO BEGIN
000136                          FOR L := 0 TO 1 DO BEGIN
000141                                DSSPTR^.REV[I*8 + J*4 + K*2 + L] := L*8 + K*4 + J*2 + I;
000155                          END
000155                    END
000160              END
000164        END;
000174       FOR I := 0 TO 127 DO BEGIN
000177              FOR J := 0 TO 7 DO BEGIN
000203                   DSSPTR^.CHARSET[I,J] := 0;
000222              END
000222       END;
000232       FOR I := 0 TO 15 DO BEGIN
000235              FOR J := 0 TO 7 DO BEGIN
000241                   DSSPTR^.BUILD[I,J] := ' ';
000253              END
000253       END;
000261         (* BUILD UP LOOKUP TABLE FOR DISPLAY TEXT *)
000263         DSSPTR^.LOOKUP[1,1] := ORD('L') + 64;DSSPTR^.LOOKUP[1,2] := ORD('N') + 64;DSSPTR^.LOOKUP[1,3] := LN;
000305         DSSPTR^.LOOKUP[2,1] := ORD('P') + 64;DSSPTR^.LOOKUP[2,2] := ORD('T') + 64;DSSPTR^.LOOKUP[2,3] := PT;
000325         DSSPTR^.LOOKUP[3,1] := ORD('C') + 64;DSSPTR^.LOOKUP[3,2] := ORD('H') + 64;DSSPTR^.LOOKUP[3,3] := CH;
000345         DSSPTR^.LOOKUP[4,1] := ORD('C') + 64;DSSPTR^.LOOKUP[4,2] := ORD('A') + 64;DSSPTR^.LOOKUP[4,3] := CA;
000365         DSSPTR^.LOOKUP[5,1] := ORD('S') + 64;DSSPTR^.LOOKUP[5,2] := ORD('C') + 64;DSSPTR^.LOOKUP[5,3] := SC;
000406         DSSPTR^.LOOKUP[6,1] := ORD('M') + 64;DSSPTR^.LOOKUP[6,2] := ORD('L') + 64;DSSPTR^.LOOKUP[6,3] := ML;
000426         DSSPTR^.LOOKUP[7,1] := ORD('M') + 64;DSSPTR^.LOOKUP[7,2] := ORD('R') + 64;DSSPTR^.LOOKUP[7,3] := MR;
000446         DSSPTR^.LOOKUP[8,1] := ORD('M') + 64;DSSPTR^.LOOKUP[8,2] := ORD('U') + 64;DSSPTR^.LOOKUP[8,3] := MU;
000467         DSSPTR^.LOOKUP[9,1] := ORD('M') + 64;DSSPTR^.LOOKUP[9,2] := ORD('B') + 64;DSSPTR^.LOOKUP[9,3] := MB;
000510         DSSPTR^.LOOKUP[10,1] := ORD('E') + 64;DSSPTR^.LOOKUP[ 10,2] := ORD('L') + 64;DSSPTR^.LOOKUP[10,3] := EL;
000530         DSSPTR^.LOOKUP[11,1] := ORD('O') + 64;DSSPTR^.LOOKUP[ 11,2] := ORD('V') + 64;DSSPTR^.LOOKUP[11,3] := OV;
000550         DSSPTR^.LOOKUP[12,1] := ORD('O') + 64;DSSPTR^.LOOKUP[ 12,2] := ORD('A') + 64;DSSPTR^.LOOKUP[12,3] := OA;
000570         DSSPTR^.LOOKUP[13,1] := ORD('U') + 64;DSSPTR^.LOOKUP[ 13,2] := ORD('N') + 64;DSSPTR^.LOOKUP[13,3] := UN;
000611         DSSPTR^.LOOKUP[14,1] := ORD('U') + 64;DSSPTR^.LOOKUP[ 14,2] := ORD('E') + 64;DSSPTR^.LOOKUP[14,3] := UE;
000631         DSSPTR^.LOOKUP[15,1] := ORD('N') + 64;DSSPTR^.LOOKUP[ 15,2] := ORD('P') + 64;DSSPTR^.LOOKUP[15,3] := NP;
000651         DSSPTR^.LOOKUP[16,1] := ORD('N') + 64;DSSPTR^.LOOKUP[ 16,2] := ORD('D') + 64;DSSPTR^.LOOKUP[16,3] := ND;
000671         DSSPTR^.LOOKUP[17,1] := ORD('R') + 64;DSSPTR^.LOOKUP[ 17,2] := ORD('P') + 64;DSSPTR^.LOOKUP[17,3] := RP;
000712         DSSPTR^.LOOKUP[18,1] := ORD('R') + 64;DSSPTR^.LOOKUP[ 18,2] := ORD('D') + 64;DSSPTR^.LOOKUP[18,3] := RD;
000732         DSSPTR^.LOOKUP[19,1] := ORD('C') + 64;DSSPTR^.LOOKUP[ 19,2] := ORD('R') + 64;DSSPTR^.LOOKUP[19,3] := CR;
000752         DSSPTR^.LOOKUP[20,1] := ORD('L') + 64;DSSPTR^.LOOKUP[ 20,2] := ORD('I') + 64;DSSPTR^.LOOKUP[20,3] := LI;
000772         DSSPTR^.LOOKUP[21,1] := ORD('A') + 64;DSSPTR^.LOOKUP[ 21,2] := ORD('L') + 64;DSSPTR^.LOOKUP[21,3] := AL;
001012         DSSPTR^.LOOKUP[22,1] := ORD('N') + 64;DSSPTR^.LOOKUP[ 22,2] := ORD('R') + 64;DSSPTR^.LOOKUP[22,3] := NR;
001032         DSSPTR^.LOOKUP[23,1] := ORD('T') + 64;DSSPTR^.LOOKUP[ 23,2] := ORD('T') + 64;DSSPTR^.LOOKUP[23,3] := TT;
0010[illegible] DSSPTR^.LOOKUP[24,1] := ORD('T') + 64;DSSPTR^.LOOKUP[ 24,2] := ORD('U') + 64;DSSPTR^.LOOKUP[24,3] := TU;
```

```
001073         DSSPTR^.LOOKUP[25,1] := ORD('T') + 64;DSSPTR^.LOOKUP[ 25,2] := ORD('E') + 64;DSSPTR^.LOOKUP[25,3] := TE;
001114         DSSPTR^.LOOKUP[26,1] := ORD('G') + 64;DSSPTR^.LOOKUP[ 26,2] := ORD('O') + 64;DSSPTR^.LOOKUP[26,3] := GO;
001134         DSSPTR^.LOOKUP[27,1] := ORD('S') + 64;DSSPTR^.LOOKUP[ 27,2] := ORD('U') + 64;DSSPTR^.LOOKUP[27,3] := SU;
001155         DSSPTR^.LOOKUP[28,1] := ORD('S') + 64;DSSPTR^.LOOKUP[ 28,2] := ORD('A') + 64;DSSPTR^.LOOKUP[28,3] := SA;
001176         DSSPTR^.LOOKUP[29,1] := ORD('U') + 64;DSSPTR^.LOOKUP[ 29,2] := ORD('U') + 64;DSSPTR^.LOOKUP[29,3] := UU;
001216         DSSPTR^.LOOKUP[30,1] := ORD('U') + 64;DSSPTR^.LOOKUP[ 30,2] := ORD('A') + 64;DSSPTR^.LOOKUP[30,3] := UA;
001236        FOR I := 32 TO 127 DO DSSPTR^.CHARS[I] := ' ';
001254        FOR I := 1 TO 26 DO DSSPTR^.CHARS[I+64] := CHR(I+ORD('A')-1);
001275        FOR I := 1 TO 10 DO DSSPTR^.CHARS[I+47] := CHR(I+ORD('0')-1);
001316        DSSPTR^.CHARS[32] := ' ';
001323        DSSPTR^.CHARS[33] := '!';
001330        DSSPTR^.CHARS[34] := '"';
001335        DSSPTR^.CHARS[35] := '#';
001342        DSSPTR^.CHARS[36] := '$';
001347        DSSPTR^.CHARS[37] := '%';
001354        DSSPTR^.CHARS[38] := '&';
001361        DSSPTR^.CHARS[39] := '''';        (* SINGLE QUOTE *)
001366        DSSPTR^.CHARS[40] := '(';
001373        DSSPTR^.CHARS[41] := ')';
001400        DSSPTR^.CHARS[42] := '*';
001405        DSSPTR^.CHARS[43] := '+';
001412        DSSPTR^.CHARS[44] := ',';
001417        DSSPTR^.CHARS[45] := '-';
001424        DSSPTR^.CHARS[46] := '.';
001431        DSSPTR^.CHARS[47] := '/';
001436        DSSPTR^.CHARS[59] := ';';
001443        DSSPTR^.CHARS[60] := '<';
001450        DSSPTR^.CHARS[61] := '=';
001455        DSSPTR^.CHARS[62] := '>';
001462        DSSPTR^.CHARS[63] := '?';
001467        DSSPTR^.CHARS[91] := '[';
001474        DSSPTR^.CHARS[92] := '\';
001501        DSSPTR^.CHARS[93] := ']';
001506        DSSPTR^.CHARS[95] := '_';     (* UNDERLINE *)
001513        FOR I := 0 TO 31 DO DSSPTR^.CHARS[I+96] := CHR(I + 32);
001534        DSSPTR^.CONV[' '] := 32;
001541        DSSPTR^.CONV['!'] := 33;
001546        DSSPTR^.CONV['"'] := 34;
001553        DSSPTR^.CONV['#'] := 35;
001560        DSSPTR^.CONV['$'] := 36;
001565        DSSPTR^.CONV['%'] := 37;
001572        DSSPTR^.CONV['&'] := 38;
001577        DSSPTR^.CONV[''''] := 39;
001604        DSSPTR^.CONV['('] := 40;
001611        DSSPTR^.CONV[')'] := 41;
001616        DSSPTR^.CONV['*'] := 42;
001623        DSSPTR^.CONV['+'] := 43;
001630        DSSPTR^.CONV[','] := 44;
001635        DSSPTR^.CONV['-'] := 45;
001642        DSSPTR^.CONV['.'] := 46;
001647        DSSPTR^.CONV['/'] := 47;
001654        DSSPTR^.CONV[';'] := 59;
001661        DSSPTR^.CONV['<'] := 60;
001666        DSSPTR^.CONV['='] := 61;
001673        DSSPTR^.CONV['>'] := 62;
001700        DSSPTR^.CONV['?'] := 63;
001705        DSSPTR^.CONV['['] := 91;
001712        DSSPTR^.CONV['\'] := 92;
001717        DSSPTR^.CONV[']'] := 93;
001724        DSSPTR^.CONV['_'] := 95;
001731          PUTCH(OUTPUT,CLEAR);    (* CLEAR SCREEN *);
001733        OPEN(ALTCHARSET,CHFILE,FALSE);    (* OPEN CHARACTER SET FILE *)
001737        RESET(ALTCHARSET);
001741        GETSET(ALTCHARSET,CHMEM,CHNO);    (* GET THE CHARACTERS *)
001744        SETCOORD(140,256,DOTS);
001750        MODE(DSPTEXT,DSPNONE,DSPNORM,DSPNO);
001756         WRITE(OUTPUT,'CYBER IS LOADING CHARACTERS');
001763        LOADCHRS(CHMEM,0,CHNO);
001766        MODE(DSPTEXT,DSPNONE,DSPNORM,DSPNO);
001774        WRITE(OUTPUT,' - DONE.');
002001        CLOSE(ALTCHARSET);
002003        PUTCH(OUTPUT,CLEAR);      (* CLEAR SCREEN *)
002005        END;                      (* ** INITARRAYS ** *)
002031
002031
002031
002031
002031
002031
```

```
002031  (*$L'PROCEDURES FOR BLOCKS'*)
002031                 (*-----------------------------------------------------
002031                 *                                                      *
002031                 =                                                      =
002031                 =                                                      =
002031                 =  BBBBBB   LL        OOOOO    CCCCC   KK   KK   SSSSSS  =
002031                 =  BB   BB  LL       OO   OO  CC   CC  KK  KK   SS       =
002031                 =  BB   BB  LL       OO   OO  CC       KK KK    SS       =
002031                 =  BBBBBB   LL       OO   OO  CC       KKKK      SSSSS   =
002031                 =  BB   BB  LL       OO   OO  CC       KKKKK         SS  =
002031                 =  BB   BB  LL       OO   OO  CC   CC  KK  KK        SS  =
002031                 =  BBBBBB   LLLLLLL   OOOOO    CCCCC   KK   KK  SSSSSS   =
002031                 =                                                      =
002031                 =                                                      =
002031                 *                                                      *
002031                 ------------------------------------------------------*)
002031
002031  (*==========================================*)
002031     PROCEDURE CREATEBLOCK (
000002                          VAR ID : INTEGER;     (* BLOCK ID, FROM 1 TO 20 *)
000003                          X1,XL : INTEGER;      (* ORIGIN AND LENGTH IN HORIZONTAL DIRECTION *)
000005                          Y1,YL : INTEGER;      (* ORIGIN AND LENGTH IN VERTICAL DIRECTION *)
000007                          M1,M2 : METRIC;  (* METRIC FOR X&Y AND XLEN&YLEN.
000011                                              THE METRICS ARE IN TERMS OF DOTS, CHARACTERS,
000011                                              AND FRACTIONS (0..100) OF A SCREEN *)
000011                          CENTERING : ADJ;      (* WHERE THE ORIGIN IS. CENTER,LL,LR,UL,UR *)
000012                          OUTLINE : INTEGER;    (* THE THICKNESS OF THE OUTLINE. + GROWS INWARD.
000013                                                  - GROWS OUTWARD. *)
000013                          OVERYFLOW : SCROLLTYPE;  (* TELLS HOW TO HANDLE END OF PAGE CONDITION.
000014                                                      PRESENTLY SCROLL, NOSCROLL *)
000014                          OVERXFLOW : WRAPTYPE;    (* TELLS HOW TO HANDLE END OF LINE.
000015                                                      EITHER WRAP OR NOWRAP. *)
000015                          CHSET,ALTCHSET : CHSETTYPE;  (* THE DEFAULT AND ALTERNATE CHARACTER
000017                                                          SETS FOR THIS BLOCK.
000017                                                          PRESENTLY STANDARD, ALTERNATE *)
000017                          VAR ERROR : ERRORTYPE    (* ERRORS.
000020                                                     BLOCKNOTNESTED,
000020                                                     TOOMANYBLOCKS,
000020                                                     BLOCKOFFSCREEN *)      );
000020
000020      (* THIS PROCEDURE CREATES VIRTUAL TERMINALS.
000020         YOU SHOULD BE ABLE TO OPERATE ON A VIRTUAL TERMINAL THE SAME WAY
000020         YOU WOULD OPERATE ON A REAL ONE. THUS, YOU MUST SPECIFY HOW
000020         TEXT BEYOND THE END OF THE VIRTUAL SCREEN IS HANDELED. THIS
000020         IS DONE BY SPECIFYING OVERFLOW. YOU MUST ALSO STATE THE SIZE
000020         AND LOCATION OF THE VIRTUAL TERMINAL. NOTICE THAT THE SCREEN
000020         AREA IS 512X512 DOTS. IF METRIC IS DOTS, THEN THE SIZE UNITS RANGE
000020         FROM (0..511). YOU MUST ALSO STATE HOW THE VIRTUAL TERMINAL WILL BE
000020         OUTLINED. NO OUTLINE AT ALL IS ACHIEVED BY SETTING OUTLINE TO 0.
000020         THE PARAMETER CENTERING SPECIFIES WHERE THE USER ORIGIN IS.
000020         YOU HAVE THE OPTION OF THE FOUR CORNERS OR THE CENTER.
000020         BLOCKS CAN BE NESTED. IN PARTICULAR BLOCK 0 IS THE FULL
000020         SCREEN. SOME OPERATIONS ON BLOCKS AFFECT ANY NESTED BLOCKS
000020         ONE SUCH OPERATION IS THE CLEAR OPERATION. TO CLEAR THE SCREEN,
000020         YOU DO THE CLEAR OPERATION ON BLOCK 0. LIKE A STANDARD TERMINAL,
000020         VIRTUAL TERMINALS HAVE STANDARD AND ALTERNATE CHARACTER
000020         SETS. ONE OF THE OPERATIONS WHEN DISPLAYING TEXT IS TO
000020          DISPLAY TEXT IN THE SPECIFIED ALTERNATE SET.
000020          *)
000020
000020      TYPE
000020          PARRAY = ARRAY[1..BLOCKMAX] OF INTEGER;
000020
000020      VAR
000020          X,XLEN,Y,YLEN : INTEGER;
000024          PCNT : INTEGER;
000025          PROGENY : PARRAY;
000051          XS,YS : INTEGER;
000053          XE,YE : INTEGER;
000055          TID,TID1 : INTEGER;
000057          INSIDE : NESTTYPE;
000060          I : INTEGER;
000061          J : INTEGER;
000062          PTX,PTY,LTX,LTY : INTEGER;
000066
000066          FUNCTION STATUS (
000002                            VAR TID : INTEGER;
000003                            ID : INTEGER;
000004                            VAR PROGENY : PARRAY;
000005                            VAR PCNT : INTEGER
000006                           ) : NESTTYPE;
000007
000007          VAR
000007              TID1 : INTEGER;
000010              XS,YS,XE,YE : INTEGER;
000014              XS1,YS1,XE1,YE1 : INTEGER;
000020
000020          BEGIN
000020              PCNT := 0;
000005              TID1 := TID;
000007              REPEAT
000007                  XS := DSSPTR^.BLK[ID].XS;
000021                  YS := DSSPTR^.BLK[ID].YS;
000030                  XE := DSSPTR^.BLK[ID].XE;
000040                  YE := DSSPTR^.BLK[ID].YE;
000050                  XS1 := DSSPTR^.BLK[TID].XS;
000062                  YS1 := DSSPTR^.BLK[TID].YS;
000072                  XE1 := DSSPTR^.BLK[TID].XE;
000102                  YE1 := DSSPTR^.BLK[TID].YE;
000112                  INSIDE := NEST(XS,XE,YS,YE,
000115                                 XS1,XE1,YS1,YE1);
000125                  TID1 := DSSPTR^.BLK[TID].NEIGHBORS;
000137                  IF INSIDE = REVNESTED THEN BEGIN
000140                      PCNT := PCNT + 1;
000142                      PROGENY[PCNT] := TID;
000146                  END;
000146              UNTIL ((TID1 = BLOCKMAX + 1) OR
000151                     (INSIDE = OVERLAP)    OR
000154                     (INSIDE = NESTED )        );
000156
000156              IF INSIDE = NESTED THEN BEGIN
000160                  IF DSSPTR^.BLK[TID].INSIDERS <> 0 THEN BEGIN
000172                      TID := DSSPTR^.BLK[TID].INSIDERS;
000202                      INSIDE := STATUS(TID,ID,PROGENY,PCNT);
```

```
000211                   IF PCNT <> 0 THEN BEGIN
000214                       STATUS := REVNESTED;
000215                   END
000215                   ELSE BEGIN
000216                       STATUS := INSIDE;    (* THIS WILL HAVE THE VALUE OVERLAP *)
000220                   END;
000220               END;
000220           END
000220           ELSE BEGIN
000221               IF PCNT <> 0 THEN BEGIN
000223                   STATUS := REVNESTED;
000224               END
000224               ELSE BEGIN
000225                   STATUS := INSIDE;
000227               END;
000227           END;
000227       END;
000262
000262   BEGIN
000262       (* LOOK FOR A PLACE *)
000262       ID := 0;
000005       I := 1;
000007       WHILE (I <= BLOCKMAX) AND (ID = 0) DO BEGIN
000014           IF NOT DSSPTR^.BLK[I].INUSE THEN BEGIN
000025               ID := I;
000026           END;
000026           I := I + 1;
000030       END;                  (* OF WHILE *)
000031       IF ID <> 0 THEN BEGIN
000033           CONVMETRIC(X1,Y1,M1,X,Y);
000040           CONVMETRIC(XL,YL,M2,XLEN,YLEN);
000046           CENTPRIT(X,XLEN,Y,YLEN,CENTERING,XS,YS);
000057           IF ((X >= 0) AND (X + XLEN <= 512) AND
000062               (Y >= 0) AND (Y + YLEN <= 512)      ) THEN BEGIN     (*IN MAIN AREA *)
000065
000065               (* CHECK FOR NESTING *)
000065               IF DSSPTR^.BLOCK0 = (BLOCKMAX + 1) THEN BEGIN
000074                   DSSPTR^.BLOCK0 := ID;
000101                   DSSPTR^.BLK[ID].NEIGHBORS := BLOCKMAX + 1;
000111                   DSSPTR^.BLK[ID].INSIDERS := BLOCKMAX + 1;
000121               END
000121               ELSE BEGIN
000122                   TID := DSSPTR^.BLOCK0;
000130                   INSIDE := STATUS(TID,ID,PROGENY,PCNT);
000137                   IF (INSIDE <> OVERLAP) OR (DSSPTR^.BLOCK0 = ID) THEN BEGIN
000147                       IF PCNT = 0 THEN BEGIN
000151
000151                           (* NEST CASE *)
000151                           TID1 := DSSPTR^.BLK[TID].INSIDERS;
000162                           DSSPTR^.BLK[TID].INSIDERS := ID;
000172                           DSSPTR^.BLK[ID].NEIGHBORS := TID1;
000202                           DSSPTR^.BLK[ID].INSIDERS := (BLOCKMAX + 1);
000213                       END
000213                       ELSE BEGIN
000214
000214                           (* REVERSE NESTING CASE *)
000214                           (* LOOP THROUGH PROGENY ARRAY *)
```

```
000214                                (* MAKE PROGENY INSIDERS *)
000214                                (* ALSO, IN THIS CASE, DON'T UPDATE TARGARRAY! *)
000214  (* FILLED IN LATER *)
000214                            END;
000214                        END;
000214                    END;
000214                    IF INSIDE <> OVERLAP THEN BEGIN
000217                        IF OUTLINE <> 0 THEN BEGIN
000221                            DRAWBOX(XS,XLEN,YS,YLEN,DOTS,DOTS,LL,OUTLINE,DSPNORM,DSPNO);
000235                        END;
000235                        DSSPTR^.BLK[ID].INUSE := TRUE;
000246                        DSSPTR^.BLK[ID].XS := XS;
000257                        DSSPTR^.BLK[ID].XE := XS + XLEN;
000267                        DSSPTR^.BLK[ID].YS := YS;
000277                        DSSPTR^.BLK[ID].YE := YS + YLEN;
000310                        DSSPTR^.BLK[ID].OVERYFLOW := OVERYFLOW;
000322                        DSSPTR^.BLK[ID].CHSET := CHSET;
000334                        DSSPTR^.BLK[ID].OUTLINE := OUTLINE;
000344                        DSSPTR^.BLK[ID].ALTCHSET := ALTCHSET;
000356                        DSSPTR^.BLK[ID].TARGS := NIL;
000366                        IF OUTLINE > 0 THEN BEGIN
000370                            DSSPTR^.BLK[ID].TEXTORGX := XS + OUTLINE + 4;
000402                            DSSPTR^.BLK[ID].TEXTORGY := DSSPTR^.BLK[ID].YE - (OUTLINE + 22);
000423                            DSSPTR^.BLK[ID].TEXTENDX := DSSPTR^.BLK[ID].XE - (OUTLINE + 12);
000443                            DSSPTR^.BLK[ID].TEXTENDY := DSSPTR^.BLK[ID].YS + OUTLINE + 6;
000463                        END
000463                        ELSE BEGIN
000464                            DSSPTR^.BLK[ID].TEXTORGX := XS + 4;
000475                            DSSPTR^.BLK[ID].TEXTORGY := DSSPTR^.BLK[ID].YE - 22;
000515                            DSSPTR^.BLK[ID].TEXTENDX := DSSPTR^.BLK[ID].XE - 12;
000534                            DSSPTR^.BLK[ID].TEXTENDY := DSSPTR^.BLK[ID].YS + 6;
000553                        END;
000553                        DSSPTR^.BLK[ID].CURLINE := 1;
000564                        DSSPTR^.BLK[ID].LINELEN := ((DSSPTR^.BLK[ID].TEXTENDX - DSSPTR^.BLK[ID].TEXTORGX) DIV 8) + 1;
000614                        DSSPTR^.BLK[ID].MAXLINE := ((DSSPTR^.BLK[ID].TEXTORGY - DSSPTR^.BLK[ID].TEXTENDY) DIV 16) + 1;
000644                        (* SETCOORD(DSSPTR^.BLK[ID].TEXTORGX,DSSPTR^.BLK[ID].TEXTORGY,DOTS); *)
000644                        DSSPTR^.BLK[ID].TEXPX := DSSPTR^.BLK[ID].TEXTORGX;
000663                        DSSPTR^.BLK[ID].TARGPX := DSSPTR^.BLK[ID].TEXPX;
000701                        DSSPTR^.BLK[ID].TEXPY := DSSPTR^.BLK[ID].TEXTORGY + 16;
000717                        DSSPTR^.BLK[ID].PLOTORGX := DSSPTR^.BLK[ID].TEXTORGX;
000736                        DSSPTR^.BLK[ID].PLOTORGY := DSSPTR^.BLK[ID].TEXTENDY;
000754                        DSSPTR^.BLK[ID].WMODE := DSPNORM;
000764                        DSSPTR^.BLK[ID].HEAD := NIL;
000773                        DSSPTR^.BLK[ID].CURPTR := NIL;
001003                        NEW(DSSPTR^.BLK[ID].TARGS);      (*ALLOCATE THE TARGET ARRAY *)
001014                        FOR I := 1 TO TARGMAX DO BEGIN
001020                            DSSPTR^.BLK[ID].TARGS^[I] := NIL;
001037                        END;
001043
001043
001043                        (* FIND ALL POSSIBLE TARGETS TO THIS BLOCK *)
001043                        FTX := DSSPTR^.BLK[ID].XS DIV 32;
001055                        FTY := DSSPTR^.BLK[ID].YS DIV 32;
001066                        LTX := (DSSPTR^.BLK[ID].XE DIV 32) - 1;
001100                        LTY := (DSSPTR^.BLK[ID].YE DIV 32) - 1;
001111                        IF (LTX >= FTX) AND (LTY >= FTY) THEN B
001113                            FOR I := FTX TO LTX DO BEGIN
```

```
001117                             FOR J := FTY TO LTY DO BEGIN
001124                                 DSSPTR^.TARGARRAY[I,J].BLOCKID := ID;
001140                             END;
001143                         END;
001152                     END;
001152
001152                     ERROR := NOERROR;
001152                 END
001152                 ELSE BEGIN
001153                     ERROR := BLOCKNOTNESTED;
001155                 END;
001155             END
001155             ELSE BEGIN
001156                 ERROR := BLOCKOFFSCREEN;
001160             END;
001160         END
001160         ELSE BEGIN
001161             (* CASE OF TOO MANY BLOCKS *)
001161             ERROR := TOOMANYBLOCKS;
001163         END;
001163     END;                            (* OF CREATEBLOCK *)
001265
001265
001265
001265
001265  (**************************************************)
001265     PROCEDURE DSTBLOCK (
000002                              ID : INTEGER;  (* WHICH BLOCK TO DESTROY*)
000003                              VAR ERROR : ERRORTYPE  (* IS SET TO NOSUCHBLOCK IF YOU
000004                                                      ATTEMPT TO DESTROY A BLOCK NOT CREATED,
000004                                                      OR THE ID IS WIERD *)      );
000004
000004     (* THIS ROUTINE DOES THE OBVIOUS TASK OF GETTING RID OF BLOCKS.
000004        ALL NESTED BLOCKS ARE DESTROYED.
000004        TO DESTROY ALL BLOCKS, DESTROY BLOCK 0 *)
000004
000004     VAR
000004         TID,TID1 : INTEGER;
000006         CONT : BOOLEAN;
000007         ID1 : INTEGER;
000010          I,J : INTEGER;
000012          XS,YS,XE,YE : INTEGER;
000016        FTX,FTY,LTX,LTY : INTEGER;
000022
000022          PROCEDURE KILLBLOCK  (
000002                                 ID : INTEGER
000003                                );
000003
000003          VAR
000003         PTR,PTR1 : DSPBUFPTR;
000005              I,J : INTEGER;
000007              TID,TID1 : INTEGER;
000011
000011          BEGIN
000011             DSSPTR^.BLK[ID].INUSE := FALSE;
000015
000015             (* DESTROY TARGETS *)
```

```
000015                FOR I := 0 TO 15 DO BEGIN
000020                    FOR J := 0 TO 15 DO BEGIN
000024                        IF DSSPTR^.TARGARRAY[I,J].TARG <> NIL THEN BEGIN
000036                            IF DSSPTR^.TARGARRAY[I,J].BLOCKID = ID THEN DSSPTR^.TARGARRAY[I,J].TARG := NIL;
000057                        END;
000057                    END;
000064                END;
000070                FOR I := 1 TO TARGMAX DO BEGIN
000073                    IF DSSPTR^.BLK[ID].TARGS^[I] <> NIL THEN BEGIN
000113                        DISPOSE(DSSPTR^.BLK[ID].TARGS^[I]);
000131                    END;
000131                END;
000136                DISPOSE(DSSPTR^.BLK[ID].TARGS);
000150
000150                (* DESTROY TEXT BUFFERS *)
000150                PTR := DSSPTR^.BLK[ID].HEAD;
000162                WHILE PTR <> NIL DO BEGIN
000165                    PTR1 := PTR;
000166                    DISPOSE(PTR1);
000170                    PTR := PTR^.NEXT;
000176                END;
000177
000177                (* REMOVE FROM DISABLED LIST *)
000177                IF DISABLED(ID,ID1) THEN DSSPTR^.DISARM[ID1] := 0;
000213
000213                (* DESTROY ALL NESTED BLOCKS *)
000213                TID := DSSPTR^.BLK[ID].INSIDERS;
000225                DSSPTR^.BLK[ID].INSIDERS := BLOCKMAX + 1;
000234                WHILE TID <> (BLOCKMAX + 1) DO BEGIN
000237                    TID1 := DSSPTR^.BLK[TID].NEIGHBORS;
000247                    KILLBLOCK(TID);
000251                    DSSPTR^.BLK[ID].NEIGHBORS := BLOCKMAX + 1;
000262                    TID := TID1;
000264                END;
000265            END;                        (* OF KILLBLOCK *)
000305
000305        BEGIN
000305            CONT := ID = 0;
000011            IF NOT CONT THEN CONT := DSSPTR^.BLK[ID].INUSE;
000024            IF CONT THEN BEGIN
000026
000026                (* CLEAR SCREEN THAT BLOCK COVERS *)
000026                IF ID = 0 THEN BEGIN
000030                    MODE(DSPTEXT,DSPNONE,DSPNORM,ISPNO);
000035                    PUTCH(OUTPUT,CLEAR);
000037                END
000037                ELSE BEGIN
000040                    IF DSSPTR^.BLK[ID].OUTLINE < 0 THEN BEGIN
000052                        XS := DSSPTR^.BLK[ID].XS + DSSPTR^.BLK[ID].OUTLINE;
000070                        YS := DSSPTR^.BLK[ID].YS + DSSPTR^.BLK[ID].OUTLINE;
000106                        XE := DSSPTR^.BLK[ID].XE - DSSPTR^.BLK[ID].OUTLINE;
000125                        YE := DSSPTR^.BLK[ID].YE - DSSPTR^.BLK[ID].OUTLINE;
000143                    END
000143                    ELSE BEGIN
000144                        XS := DSSPTR^.BLK[ID].XS;
000154                        YS := DSSPTR^.BLK[ID].YS;
000163                        XE := DSSPTR^.BLK[ID].XE;
```

```
000173                        YE := DSSPTR^.BLK[ID].YE;
000203                     END;
000203                     FOR I := XS TO XE DO BEGIN
000210                        SETCOORD(I,YS,DOTS);
000214                        DRAWLINE(I,YE,DOTS,ERASE,DSPNO);
000222                     END;
000227                  END;
000227
000227                  (* KILL DECEDANTS *)
000227                  IF ID <> 0 THEN BEGIN
000231                     KILLBLOCK(ID);
000233
000233                     (* FIND ALL POSSIBLE TARGETS TO THIS BLOCK *)
000233                     FTX := DSSPTR^.BLK[ID].XS DIV 32;
000245                     FTY := DSSPTR^.BLK[ID].YS DIV 32;
000256                     LTX := (DSSPTR^.BLK[ID].XE DIV 32) - 1;
000270                     LTY := (DSSPTR^.BLK[ID].YE DIV 32) - 1;
000301                     IF (LTX >= FTX) AND (LTY >= FTY) THEN BEGIN
000303                        FOR I := FTX TO LTX DO BEGIN
000307                           FOR J := FTY TO LTY DO BEGIN
000314                              DSSPTR^.TARGARRAY[I,J].BLOCKID := 0;
000325                           END;
000331                        END;
000336                     END;
000336                  END
000336                  ELSE BEGIN
000337                     TID := DSSPTR^.BLOCK0;
000345                     DSSPTR^.BLOCK0 := BLOCKMAX + 1;
000352                     WHILE TID <> BLOCKMAX + 1 DO BEGIN
000355                        TID1 := DSSPTR^.BLK[TID].NEIGHBORS;
000365                        KILLBLOCK(TID);
000367                        DSSPTR^.BLK[TID].NEIGHBORS := BLOCKMAX + 1;
000400                        TID := TID1;
000402                     END;
000403
000403                     (* RESET BLOCK ID IN TARGARRAY *)
000403                     FOR I := 0 TO 15 DO BEGIN
000407                        FOR J := 0 TO 15 DO BEGIN
000413                           DSSPTR^.TARGARRAY[I,J].BLOCKID := 0;
000424                        END;
000430                     END;
000434                  END;
000434                  ERROR := NOERROR;
000436               END
000436               ELSE BEGIN
000437                  ERROR := NOSUCHBLOCK;
000441               END;
000441            END;                           (* OF DESTROY BLOCK *)
000503
000503
000503
000503
000503    (*==============================================*)
000503            PROCEDURE CLEARBLOCK (
000002                                 ID : INTEGER;    (* BLOCK ID *)
000003                                 VAR ERROR : ERRORTYPE    (* NOSUCHBLOCK *) );
000003
```

```
000004     (* THIS PROCEDURE ERASES THE SCREEN OF THE VIRTUAL TERMINAL.
000004        THIS OPERATION IS ALLOWED ON BLOCK 0. ALL NESTED BLOCKS
000004        ARE AFFECTED. *)
000004     BEGIN
000004     END;
000012
000012
000012 (*==============================================*)
000012     PROCEDURE UNDOBLOCK (
000002                        ID : INTEGER;      (* BLOCK ID *)
000003                        VAR ERROR : ERRORTYPE     (* NOSUCHBLOCK *) );
000004
000004     (* THIS IS FOR A LESS HARSH CLEANUP OF A BLOCK. THIS DOES
000004        NOT AFFECT THE NESTED BLOCKS. YOU CAN USE THIS TO SCRAP
000004        A BLOCK AND READY IT FOR NEW TEXT W/O ACTUALLY DESTROYING
000004        IT. *)
000004
000004     VAR
000004         PTR1,PTR2 : TEXTPTR;
000010         PX,PY : INTEGER;
000012
000012     BEGIN
000012         IF DSSPTR^.BLK[ID].INUSE THEN BEGIN
000015              ERROR := NOERROR;
000016              PTR1.POS := DSSPTR^.BLK[ID].HEAD^.POS;
000034              PTR1.BUF := DSSPTR^.BLK[ID].HEAD;
000044              PY := DSSPTR^.BLK[ID].TEXTORGY;
000055              PX := DSSPTR^.BLK[ID].TEXTORGX;
000065              IF PTR1.BUF <> NIL THEN BEGIN
000067                   DSPLINE(ID,PTR1,PTR2,UNDO,PX,PY);
000075                   WHILE PTR2.BUF <> NIL DO BEGIN
000100                        PTR1 := PTR2;
000102                        PY := PY  - 16;
000104                        PX := DSSPTR^.BLK[ID].TEXTORGX;
000115                        DSPLINE(ID,PTR1,PTR2,UNDO,PX,PY);
000123
000123                        (*SCRAP TEXT BUFFERS *)
000123                        IF PTR1.BUF <> PTR2.BUF THEN DISPOSE(PTR1);
000127                   END;
000130              END;
000130         END;
000130     END;
000142
000142
000142 (*==============================================*)
000142     PROCEDURE REDOBLOCK (
000002                        ID : INTEGER;      (* BLOCK ID *)
000003                        VAR ERROR : ERRORTYPE     (* NOSUCHBLOCK *) );
000004
000004     (* THIS PROCEDURE ATTEMPTS TO RECONSTRUCT THE STUFF IN A BLOCK
000004        THIS PROCEDURE IS MENT TO BE USED IF YOU GET TRANSMISSION
000004        ERRORS. *)
000004     BEGIN
000004     END;
000012
000012
000012 (*==============================================*)
```

```
000012    PROCEDURE DISARMBLOCK (
000002                                ID : INTEGER;    (* BLOCK ID *)
000003                                VAR ERROR : ERRORTYPE    (* NOSUCHBLOCK *) );
000004
000004    (* THIS PROCEDURE IS FOR DISARMING ALL THE TOUCH TARGETS
000004       IN A BLOCK. GOOD FOR MULTIPLE QUESTIONS ON ONE SCREEN *)
000004
000004    VAR
000004        I : INTEGER;
000005
000005    BEGIN
000005        IF DSSPTR^.BLK[ID].INUSE THEN BEGIN
000015             I := 1;
000017             WHILE DSSPTR^.DISARM[I] <> 0 DO BEGIN
000030                 I := I + 1;
000031             END;
000032             DSSPTR^.DISARM[I] := ID;
000041             ERROR := NOERROR;
000042        END
000042        ELSE BEGIN
000043             ERROR := NOSUCHBLOCK;
000045        END;
000045    END;
000055
000055
000055
000055
000055 (**********************************************)
000055    PROCEDURE REARMBLOCK (
000002                               ID : INTEGER;     (* BLOCK ID *)
000003                               VAR ERROR : ERRORTYPE    (* NOSUCHBLOCK *) );
000004
000004    (* THIS PROCEDURE REARMS BLOCKS. VISUALLY, THIS MEANS
000004       MAKING TOUCHED TARGETS LOOK LIKE NEW. *)
000004
000004    VAR
000004        I : INTEGER;
000005        MORE : BOOLEAN;
000006
000006    BEGIN
000006        MORE := TRUE;
000006        I := 1;
000007        IF DSSPTR^.BLK[ID].INUSE THEN BEGIN
000017             WHILE (I <= BLOCKMAX) AND MORE DO BEGIN
000022                 MORE := DSSPTR^.DISARM[I] <> ID;
000034                 I := I + 1;
000035             END;
000036             IF I <> 0 THEN DSSPTR^.DISARM[I] := 0;
000046
000046             (* MAKE TTS LOOK LIKE NEW *)
000046        END
000046        ELSE BEGIN
000047             ERROR := NOSUCHBLOCK;
000051        END;
000051    END;
000063
000063
```

```
000063  3019
000063  3020
```

```
000063  (*$L'PROCEDURES FOR TEXT'*)
000063                      (*-----------------------------------------------------------
000063                      =                                                           =
000063                      =                                                           =
000063                      =                                                           =
000063                      =                                                           =
000063                      =       TTTTTT   EEEEEEE  XX    XX  TTTTTT                  =
000063                      =         TT     EE        XX XX      TT                    =
000063                      =         TT     EE         XXX       TT                    =
000063                      =         TT     EEEEE      XXX       TT                    =
000063                      =         TT     EE         XXX       TT                    =
000063                      =         TT     EE        XX XX      TT                    =
000063                      =         TT     EEEEEEE  XX    XX    TT                    =
000063                      =                                                           =
000063                      =                                                           =
000063                      =                                                           =
000063                      ------------------------------------------------------------*)
000063
000063
000063  (*************************************************************)
000063     PROCEDURE DSTEXT (
000002                              ID : INTEGER;    (* ID OF BLOCK *)
000003                              VAR INTEXT : TEXT;  (* FILE OF INPUT TEXT *)
000004                              VAR ERROR : ERRORTYPE    (* ERRORS.
000005                                                    OUTOFBOUNDS,
000005                                                    NOMEMORY,
000005                                                    LABELTOOLONG,
000005                                                    COMMANDERROR,
000005                                                    NOSUCHBLOCK *)      );
000005
000005    (* THIS ROUTINE IS HOW YOU FILL A VIRTUAL TERMINAL WITH STUFF.
000005       YOU FIRST WRITE TEXT TO INTEXT, THEN YOU PASS THIS TEXTFILE
000005       TO THIS DISPLAY ROUTINE. JUST AS IN A REAL TERMINAL WITH
000005       GRAPHICS CAPABILITY, THERE ARE ESCAPE SEQUENCES TO DO GRAPHICS.
000005       THERE IS ALSO A SET OF ESCAPE SEQUENCES FOR PLACEMENT OF TEXT AS
000005       WELL AS THE CREATION OF TOUCH TARGETS. ALSO SUPPORTED ARE ALTERNATE
000005       CHARACTER SETS. BELOW IS THE INITIAL SET OF COMMANDS:
000005
000005       GRAPHICS
000005       --------
000005       \GOX Y SET GRAPHICS ORIGIN X DOTS ACCROSS AND Y DOTS AWAY FROM
000005              BLOCK ORIGIN.
000005       \SCX Y SET CURSOR X DOTS HORIZONTALLY AND Y DOTS
000005              VERTICALLY FROM THE TEXT ORIGIN. IF X OR Y ARE THE CHARACTERS
000005              "999" THEN THE CURRENT POSITION IS USED.
000005       \LNX Y DRAW A LINE FROM THE LAST POSITION TO (X,Y).
000005       \PTX Y DRAW A POINT AT (X,Y).
000005       \CH# X Y DRAW A CHARACTER WHOSE ORD IS # AT (X,Y). (NORMAL SET)
000005       \CA# X Y DRAW A CHARACTER WHOSE ORD IS # AT (X,Y). (ALTERNATE SET)
000005
000005       TEXT
000005       ----
000005       \MLX    SET LEFT MARGIN X CHARCTERS AWAY FROM TEXT ORIGIN.
000005       \MRX    SET RIGHT MARGIN X CHARCTERS AWAY FROM TEXT ORIGIN.
000005       \MUY    SET UPPER MARGIN.
000005       \MBY    SET BOTTOM MARGIN.
000005       \OV$    OVERSTRIKE THE LAST CHARACTER WITH THE CHARACTER $
000005          \OA#   OVERSTRIKE THE LAST CHARACTER WITH THE ALTERNATE CHARACTER SET
000005                 CHARACTER WHOSE ORD IS #
000005          \SU$   SUBSCRIPT THE CHARACTER $.
000005          \SA#   SUBSCRIPT THE ALTERNATE CHARACTER WHOSE ORD IS #.
000005          \UU$   SUPERSCRIPT THE CHARACTER $.
000005          \UA#   SUPERSCRIPT THE ALTERNATE CHARACTER WHOSE ORD IS #.
000005          \ULTEXT\UE UNDERLINE TEXT BETWEEN \UL & \UE
000005          \NP,\ND,\RP,\RD SET THE MODE OF DISPLAY.
000005                            N IS NORMAL VIDEO
000005                            R IS REVERSE VIDEO
000005                            P IS PROTECTED (I.E. OVERSTRIKE)
000005                            D IS DESTRUCTIVE
000005          \CR    CARRIAGE RETURN. THIS IS GOOD FOR DOING FANCY STUFF WITH OVERSTRIKES.
000005          \LF    LINE FEED. GO DOWN A LINE W/O A RETURN.
000005          \EL    END OF LINE.
000005          \AL#####S\NR ESCAPE TO ALTERNATE CHARACTER SET. SINCE THE ALTERNATE
000005                       SET IS MAPPED ONTO ALL THE ASCII SET, YOU REPRESENT THESE
000005                       CHARACTERS BY THEIR ORDS (SEPERATED BY SPACES, BASE 10).
000005
000005          TOUCH TARGETS
000005          ----- -------
000005
000005          \TT#LABEL\TE CREATE A TOUCH TARGET WHOSE TARGET ID IS # WITH LABEL
000005                       INSIDE IT. THIS TARGET GETS FAT AFTER GETTING TOUCHED.
000005          \TU#LABEL\TE CREATE A TOUCH TARGET WHICH TURNS INTO THE UNDERLINED
000005                       LABEL AFTER BEING TOUCHED.
000005
000005          (NOTE: \ IS THE ESCAPE CHARACTER AND IS AVAILABLE TO YOU IN THE
000005                 CONSTANT ESCAPE)
000005          *)
000005
000005
000005        VAR
000005           INTREP : CRNG;
000006           I : INTEGER;
000007           CUR : CHAR;
000010
000010
000010
000010         FUNCTION CNVT1(VAR INTEXT : TEXT;   (*TEXT FILE THAT IS BEING CONVERTED *)
000003                        VAR ERROR : ERRORTYPE    (* POSSIBLE ERROR IS COMMAND ERROR *)
000004                        ) : INTEGER;
000005
000005          (* THIS FUNCTION IS USED TO LOOK UP THE COMMAND FOR SOME
000005             DISPLAY ACTION. IT CONVERTS IT TO AN INTEGER >127 WHICH
000005             IS USED AS REPRESENTING THE COMMAND IN THE INTERNAL
000005             DISPLAY BUFFERS IN THE PROCEDURE DSPTEXT *)
000005
000005          (* USES LOOKUP = PACKED ARRAY[1..NCOMMANDS,1..3] OF 0..511 AND
000005             THE CONSTANT NCOMMANDS= NUMBER OF COMMANDS                     *)
000005
000005          VAR
000005             CH : INTEGER;
000006             CH1 : INTEGER;
000007             CHA : CHAR;
000010             CH1A : CHAR;
```

```
000011          I : INTEGER;
000012          NOTFOUND : BOOLEAN;
000013
000013        BEGIN
000013          NOTFOUND := TRUE;
000006          READ(INTEXT,CHA);
000014          CH := CNVT(INTEXT,CHA);
000020          READ(INTEXT,CH1A);
000030          CH1 := CNVT(INTEXT,CH1A);
000034          (*DO LOOKUP *)
000034          I := 1;
000036          WHILE (I <= NCOMMANDS) AND NOTFOUND DO BEGIN
000041             IF ( (DSSPTR^.LOOKUP[I,1] = CH) AND
000052                  (DSSPTR^.LOOKUP[I,2] = CH1)   ) THEN NOTFOUND := FALSE;
000064             I := I + 1;
000066          END;             (* OF WHILE *)
000067          IF NOTFOUND THEN BEGIN
000070             ERROR := COMMANDERROR;
000072             CNVT1 := 0;       (* AN ASCII NULL *)
000073          END
000073          ELSE BEGIN
000074             CNVT1 := DSSPTR^.LOOKUP[I-1,3];
000105          END;
000105        END;                    (*OF CNVT1*)
000126
000126
000126        PROCEDURE PUTICH (
000002                      CH : CRNG ;      (* INTEGER REPRESENTATION OF CHAR *)
000003                      ID : INTEGER     (* BLOCK WHOSE BUFFERS TO STORE IN *)
000004                     );
000004
000004        (* THIS PROCEDURE STUFFS THE INTEGER REPRESENTATION OF A
000004        CHARACTER IN THE TEXT BUFFERS OF BLOCK <ID>. THESE BUFFERS
000004        ARE FIXED LENGTH ARRAYS & ARE ALLOCATED INCREMENTALLY AS
000004        THEY ARE FILLED UP.
000004        *)
000004
000004        VAR
000004           NEWBUF : DSPBUFPTR;
000005
000005
000005        BEGIN
000005           IF DSSPTR^.BLK[ID].CURPTR = NIL THEN BEGIN
000016
000016              (*CASE OF NO BUFFERS AT ALL IN CURRENT USE *)
000016              (* CURPTR IS SET TO NIL IN CREATEBLOCK *)
000016              NEW(DSSPTR^.BLK[ID].CURPTR);
000026              DSSPTR^.BLK[ID].CURPTR^.NEXT := NIL;
000043              DSSPTR^.BLK[ID].CURPTR^.POS := 0;
000057              DSSPTR^.BLK[ID].CURPTR^.EPOS := 0;
000072              DSSPTR^.BLK[ID].HEAD := DSSPTR^.BLK[ID].CURPTR;
000110              DSSPTR^.BLK[ID].LSTPTR.BUF := DSSPTR^.BLK[ID].CURPTR;
000126              DSSPTR^.BLK[ID].LSTPTR.POS := DSSPTR^.BLK[ID].CURPTR^.POS;
000151           END;
000151           DSSPTR^.BLK[ID].CURPTR^.EPOS := DSSPTR^.BLK[ID].CURPTR^.EPOS + 1;
000203           IF DSSPTR^.BLK[ID].CURPTR^.EPOS > TEXBUFLEN THEN BEGIN
000217              DSSPTR^.BLK[ID].CURPTR^.EPOS := DSSPTR^.BLK[ID].CURPTR^.EPOS - 1;
```

```
000250                  (* TIME FOR A NEW BUFFER *)
000250                  NEW(NEWBUF);
000252                  DSSPTR^.BLK[ID].CURPTR^.NEXT := NEWBUF;
000267                  NEWBUF^.NEXT := NIL;
000274                  DSSPTR^.BLK[ID].CURPTR := NEWBUF;    (* POINT TO NEW BUFFER *)
000304                  DSSPTR^.BLK[ID].CURPTR^.POS := 0;
000320                  DSSPTR^.BLK[ID].CURPTR^.EPOS := 1;
000334              END;
000334
000334              (* PLACE INTEGER REPRESENTATION IN BUFFER *)
000334              DSSPTR^.BLK[ID].CURPTR^.AR[DSSPTR^.BLK[ID].CURPTR^.EPOS] := CH;
000376          END;                    ,  (* OF PUTICH *)
000406
000406
000406
000416
000416          PROCEDURE INTERPRET (
000002                                  INTREP : CRNG ;  (* INTEGER REPRESENTATION OF CHARACTER *)
000003                                  ID : INTEGER;    (* BLOCK ID *)
000004                                  ERROR : ERRORTYPE
000005                              );
000005
000005          (* THIS PROCEDURE IS FOR THE INITIAL CONVERSION OF NUMBERS
000005             IN THE TEXTFILE BEING READ IN TO THEIR INTEGER REPRESENTATION.
000005             IT ALSO DOES END OF LINE HANDLEING *)
000005
000005          VAR
000005              PTR,PTR1 : TEXTPTR;
000011              X,Y,NUM : INTEGER;
000014              CH1 : CHAR;
000015              CHO : INTEGER;
000016              SCRATCH : TEXT;
000052
000052          BEGIN
000052              PUTICH(INTREP,ID);
000015              CASE INTREP OF
000017
000017                  SC,IT,PT,GO : BEGIN      (* DOUBLE INTEGERS *)
000017                      READ(INTEXT,X,Y);
000027                      PUTICH(X,ID);
000034                      PUTICH(Y,ID);
000041                  END;
000042
000042                  CH,CA : BEGIN   (*TRIPLE INTEGERS*)
000042                      READ(INTEXT,CHO,X,Y);
000055                      PUTICH(CHO,ID);
000062                      PUTICH(X,ID);
000067                      PUTICH(Y,ID);
000074                  END;
000075
000075                  ML,MR,MU,MB,OA : BEGIN  (* SINGLE INTEGERS *)
000075                      READ(INTEXT,NUM);
000102                      PUTICH(NUM,ID);
000107                  END;
000110
000110                  AL : BEGIN      (* N INTEGERS *)
000110                      REWRITE(SCRATCH);
```

```
000112                  READ(INTEXT,CH1);
000122                  WHILE CH1 <> ESCAPE DO BEGIN
000125                      WRITE(SCRATCH,CH1);
000133                      READ(INTEXT,CH1);
000143                  END;         (* OF WHILE *)
000144                  WRITELN(SCRATCH);
000146                  RESET(SCRATCH);
000150                  REPEAT
000150                      READ(SCRATCH,NUM);
000153                      PUTICH(NUM,ID);
000160                  UNTIL EOLN(SCRATCH);
000163
000163                  (* GET REST OF END DELIMITER *)
000163                  INTREP := CNVT1(INTEXT,ERROR);
000172                  IF ERROR = NOERROR THEN BEGIN
000174                      INTERPRET(INTREP,ID,ERROR);      (* CALL ME AGAIN ! *)
000201                  END
000201                  ELSE BEGIN
000202                      PUTICH(NB,ID);  (* PUT DELIMITER IN ANYWAY *)
000206                  END;
000206              END;            (* OF N NUMBERS CASE *)
000207
000207              EL,LI : BEGIN   (* NEXT LINE CASES *)
000207                  HANDLEEOL(ID);
000211                  PTR.BUF := DSSPTR^.BLK[ID].LSTPTR.BUF;
000223                  PTR.POS := DSSPTR^.BLK[ID].LSTPTR.POS;
000234                  DSPLINE(ID,PTR,PTR1,PRINT,DSSPTR^.BLK[ID].TEXPX,DSSPTR^.BLK[ID].TEXPY);
000260                  WHILE (PTR1.BUF <> DSSPTR^.BLK[ID].CURPTR) AND
000272                         (PTR1.POS <> DSSPTR^.BLK[ID].CURPTR^.PPOS) DO BEGIN
000307                      PTR := PTR1;
000311                      HANDLEEOL(ID);
000313                      DSPLINE(ID,PTR,PTR1,PRINT,DSSPTR^.BLK[ID].TEXPX,DSSPTR^.BLK[ID].TEXPY);
000340                  END;
000341                  DSSPTR^.BLK[ID].LSTPTR := PTR1;
000352              END;            (* OF NEXT LINE CASES *)
000353
000353              TT,TU : BEGIN   (* TOUCH TARGET CASES *)
000353
000353                  (* DIVERT LABEL TO LABEL BUFFER *)
000353                  DSSPTR^.LABCTR := DSSPTR^.LABCTR + 1;
000367                  IF DSSPTR^.LABCTR <= TARGMAX THEN BEGIN      (* ONLY SO MUCH ROOM *)
000374                      NEW(DSSPTR^.LABARR[DSSPTR^.LABCTR]);     (* GET A NEW LABEL BUFFER *)
000407                      READ(INTEXT,NUM);   (* USER'S TARGET ID *)
000414                      PUTICH(NUM,ID);
000421                      PUTICH(DSSPTR^.LABCTR,ID);  (* STORE INDEX TO LABEL *)
000432                      I:=1;
000434                      READ(INTEXT,CH1);
000443                      WHILE (CH1 <> ESCAPE) AND ( I <= LABELLEN ) DO BEGIN
000450                          DSSPTR^.LABARR[DSSPTR^.LABCTR]^[I] := CH1;
000503                          I := I + 1;
000504                          READ(INTEXT,CH1);
000514                      END;
000515                      PUTICH(I-1,ID);       (* NUMBER OF CHARACTERS IN BUFFER *)
000522                      IF CH1 <> ESCAPE THEN BEGIN      (* THROW AWAY REST IF BUFFER IS TOO FULL *)
000525                          ERROR := LABELTOOLONG;
000526                          READ(INTEXT,CH1);
000536                          WHILE CH1 <> ESCAPE DO BEGIN
```

```
000541                                 READ(INTEXT,CH1);
000550                              END;     (* OF WHILE *)
000551                         END;
000551                         INTREP := CNVT1(INTEXT,ERROR);
000560                         IF ERROR = NOERROR THEN BEGIN
000562                             INTERPRET(INTREP,ID,ERROR);
000567                         END
000567                         ELSE BEGIN
000570                           PUTICH(TE,ID);
000574                         END;
000574                    END
000574                    ELSE BEGIN
000575                         ERROR := NOMEMORY;
000577                    END;
000577               END;
000600
000600               OTHERWISE        (* NOTHING *)
000621
000621
000621          END;                  (* OF CASE *)
000621
000621     END;                       (* OF INTERPRET *)
000645
000645
000645
000645 BEGIN                          (*OF MAIN PART *)
000645     IF DSSPTR^.BLK[ID].INUSE THEN BEGIN
000015          DSSPTR^.LABCTR := 0;
000022          READ(INTEXT,CHR);
000031          WHILE (EOLN(INTEXT)) DO BEGIN   (* OMIT EMPTY LINES *)
000033               READLN(INTEXT);
000036          END;
000037
000037          (* GET STUFF UNTIL END OF FILE *)
000037          WHILE (NOT EOF(INTEXT)) DO BEGIN
000041               INTREP := CNVT(INTEXT,CHR);
000047               IF INTREP = ESCCH THEN BEGIN    (* WAS A COMMAND, SO LOOK IT UP *)
000051                    INTREP := CNVT1(INTEXT,ERROR);
000056                    IF ERROR = NOERROR THEN BEGIN   (* DON'T CONTINUE IF COMMAND WAS FUNNY *)
000060                         INTERPRET(INTREP,ID,ERROR);     (* DO SOME INITIAL SEMANTICS *)
000065                    END;
000065               END
000065               ELSE BEGIN
000066                    PUTICH(INTREP,ID);
000072               END;
000072               WHILE(EOLN(INTEXT) AND (NOT EOF(INTEXT))) DO BEGIN
000075                    READLN(INTEXT);
000077               END;
000100               IF NOT EOF(INTEXT) THEN READ(INTEXT,CHR);
000110          END;
000111
000111          (* SCRATCH BUFFERS *)
000111          FOR I := 1 TO DSSPTR^.LABCTR DO BEGIN
000122               DISPOSE(DSSPTR^.LABARR[I]);
000133          END;
000140     END
000140     ELSE BEGIN
```

```
000141  3363            ERROR := NOSUCHBLOCK;
000143  3364        END;
000143  3365    END;
000157  3366
```

```
000157  (*$L'PROCEDURES FOR TOUCH TARGETS'*)
000157                   (*------------------------------------------------------------------
000157                   =                                                                  =
000157                   =                                                                  =
000157                   =           TTTTTT    00000   UU   UU   CCCCC   HH   HH            =
000157                   =             TT     00   00  UU   UU  CC   CC  HH   HH            =
000157                   =             TT     00   00  UU   UU  CC       HH   HH            =
000157                   =             TT     00   00  UU   UU  CC       HHHHHHH            =
000157                   =             TT     00   00  UU   UU  CC       HH   HH            =
000157                   =             TT     00   00  UU   UU  CC   CC  HH   HH            =
000157                   =             TT      00000    UUUUU    CCCCC   HH   HH            =
000157                   =                                                                  =
000157                   =                                                                  =
000157                   =                                                                  =
000157                   =                                                                  =
000157                   = TTTTTT    AAAAA    RRRRRR    GGGGG    EEEEEEE  TTTTTT            =
000157                   =   TT     AA   AA   RR   RR  GG   GG   EE         TT              =
000157                   =   TT     AA   AA   RR   RR  GG        EE         TT              =
000157                   =   TT     AAAAAAA   RRRRRR   GG        EEEEE      TT              =
000157                   =   TT     AA   AA   RRRR     GG  GGG   EE         TT              =
000157                   =   TT     AA   AA   RR RR    GG   GG   EE         TT              =
000157                   =   TT     AA   AA   RR  RR    GGGGG    EEEEEEE    TT              =
000157                   =                                                                  =
000157                   =                                                                  =
000157                   =-----------------------------------------------------------------*)
000157
000157
000157
000157  (*=================================================*)
000157     PROCEDURE GETTOUCHINP (*
000010                              VAR X,Y : INTEGER;*)     (* X,Y COORDINATES OF TOUCH *)
000010                              (*VAR CBUF : INBUF;*) (* ARRAY OF INPUT CHARACTERS *)
000010                              (*VAR LEN : INTEGER;*) (* LENGTH OF STRING IN CHARRAY *)
000010                              (*MKCIRCLE : BOOLEAN;*) (* MAKE A CIRCLE
000010                                                   WHERE USER TOUCHED *)
000010                              (*VAR ERROR : ERRORTYPE*) (* BADTOUCH *)(* ); *) ;
000010
000010     (* USER DESCRIPTION:
000010        THIS PROCEDURE IS PROVIDED FOR ACCEPTING TOUCH INPUT. WHEN THE
000010        SCREEN IS TOUCHED, A SMALL CIRCLE WILL BE PLACED AT THE TOUCH POINT,
000010        IF THE FLAG, MKCIRCLE, IS SET TO TRUE.
000010        IF THE INPUT WAS GARBAGE, ERROR IS SET TO BADTOUCH
000010        *)
000010
000010     (* INTERNAL DESCRIPTION:
000010        FIRST GET TERMINAL IN TOUCH MODE THEN DECODE THE INPUT.
000010        THE INPUT FROM THE TOUCH PANEL IS IN THE FORM OF A 4
000010        CHARACTER SEQUENCE WHERE:
000010
000010           CHARACTER 1 : /0000010/ STX CHARACTER
000010           CHARACTER 2 : /10/X2X3X4/10/
000010           CHARACTER 3 : /01/Y1Y2Y3Y4X1/
000010           CHARACTER 4 : /0001101/ CR CHARACTER
000010
000010        THE PURPOSE OF THE ORION IN SENDING THE STX CHARACTER
000010        IS THAT IF IT IS ECHOED BACK TO THE ORION, IT
000010        CAUSES THE ORION'S KEYBOARD TO BE DISABLED.
```

```
000010          THE CR ALLOWS THE CYBER TO KNOW THAT THIS IS THE END
000010          OF THE TOUCH INPUT SEQUENCE. NOTICE THAT THE BITS ARE
000010          BACKWARDS IN THE CHARACTERS SENT. (MSB IS LOWER ORDER)
000010          THIS IS WHY THERE IS A BIT REVERSAL LOOKUP TABLE USED
000010          IN THIS PROCEDURE. THIS ROUTINE IGNORES ANY RESPONSE
000010          THAT CONSIST OF JUST A CR. I DON'T BELIEVE THAT I REALLY
000010          NEED TO SEND OUT THE COLON E CONTROL BYTE. (THE FUNCTIONALITY
000010          OF WHICH CAN BE FOUND IN APPENDIX C OF THE NOS TIME
000010          SHARING MANUAL. THIS PROCEDURE DOES NOT WORK PERFECTLY.
000010          IF THE PROGRAM SEEMS TO MISS THE TOUCH INPUT YOU SHOULD
000010          HIT THE CR KEY AND REENTER YOUR TOUCH RESPONSE. THE REASON
000010          FOR FAULTY OPERATION PROBABLY IS DUE TO THE FACT THAT
000010          THERE IS NO TYPE-AHEAD IN THE NOS SYSTEM.
000010       *)
000010
000010       LABEL 1,2;
000010       VAR
000010           IND : INTEGER;
000011           DONE : BOOLEAN;
000012           N1 , N2 , N3 : INTEGER;
000015
000015       BEGIN
000015           SETCOORD(10,10,DOTS);
000010           MODE(DSPTEXT,TOUCH,DSPNORM,DSPNO);
000015           REPEAT
000015               PUTCH(OUTPUT,TCH);   (*GET IT IN TOUCH MODE*)
000017               WRITELN;
000021               WRITELN(':E');       (*SET IN ASCII MODE*)
000027               GETSEG(INPUT);
000031               GET(INPUT);
000035           UNTIL NOT EOS(INPUT);
000037           GETLN(INPUT,CBUF,LEN,DONE);
000042           IND := 1;                (* ERROR *)
000044           IF IND > LEN THEN GOTO 1;
000046           N1 := CNVTA(CBUF,IND);
000052           IND := IND + 1;
000054           IF (IND > LEN) OR (N1 <> STX) THEN GOTO 1;  (* ERROR *)
000060           N2 := CNVTA(CBUF,IND);
000064           IND := IND + 1;
000066           IF IND > LEN THEN GOTO 1;    (* ERROR *)
000071           N3 := CNVTA(CBUF,IND);
000075           IF IND <> LEN THEN GOTO 1;   (* ERROR *)
000100           X := ((N2 DIV 4) MOD 8) ;
000104           X := X + (N3 MOD 2) * 8;
000107           Y := ((N3 DIV 2) MOD 16) ;
000112           X := DSSPTR^.REV[X]*32 + 15;     (*REVERSE THE BITS*)
000122           Y := DSSPTR^.REV[Y]*32 + 15;
000132           IF MKCIRCLE THEN DRAWCHAR(111,X,Y,DOTS,CENTER,NORMAL,DSPNORM,DSPNO);
000144           ERROR := NOERROR;
000146           GOTO 2;                  (* ERROR ISLAND CREATION *)
000147    1:         ERROR := BADTOUCH;
000151    2: END;
000200
000200
000200 (*=============================================*)
000200       PROCEDURE GETTARGINP (
000002                               VAR ID : INTEGER;    (* BLOCK ID *)
000003                            VAR TARGID : INTEGER;   (* TARGET ID *)
000004                            CHARRAY : INBUF;     (* ARRAY OF INPUT CHARACTERS *)
000005                            CHLEN : INTEGER;     (* LENGTH OF STRING IN CHARRAY *)
000006                            INCLUDEBLOCKS : BOOLEAN;     (* INCLUDE BLOCKS AS TARGETS
000007                                                          HAS A TARGET ID OF 0 *)
000007                            ERROR : ERRORTYPE   (* MISTOUCH, BADTOUCH *)  );
000026
000026    (* USER DESCRIPTION:
000026       READS TOUCH TARGET INPUT. EACH TARGET ON THE SCREEN IS KNOWN BY A
000026       (ID,TARGID) PAIR. IF A TOUCH WAS IN AN INSENSITIVE AREA, THEN ERROR
000026       IS SET TO MISTOUCH. IF THE TOUCH WAS NOT DECODEABLE AS BEING A TOUCH,
000026       THEN ERROR IS SET TO BADTOUCH (I.E. USER PROBABLY USED THE KEYBOARD).
000026       FOR THIS CASE, THE LINE OF INPUT TEXT IS PASSED BACK TO YOU. YOU CAN
000026       THEN USE THIS ARRAY TO FETCH ANY KEYBOARD INPUT.
000026       *)
000026
000026    (* INTERNAL DESCRIPTION:
000026       CALL GETTOUCHINP, CHECK TO SEE IF WE ARE IN AN
000026       ACTIVE TARGET, DO VISUAL RESPONSE TO TOUCH ACTIONS.
000026       IF INCLUDEBLOCKS IS TRUE, THEN ANY TOUCH [illegible]
000026       INCLUDED IN A BLOCK TOUCHED WILL COME BACK [illegible]
000026       BLOCK TOUCHED AS THE BLOCKID AND A TARGID OF [illegible]
000026    *)
000026
000026
000026
000026    VAR
000026        ID1 : INTEGER;
000027        X1,Y1 : INTEGER;
000031        I : INTEGER;
000032        PTR : TPTR;
000033        X,Y : INTEGER;
000035
000035    BEGIN
000035        GETTOUCHINP(X,Y,CHARRAY,CHLEN,FALSE,ERROR);
000017        IF ERROR = NOERROR THEN BEGIN

000021            (* CONVERT TO TARG COORDS *)
000021            X1 := X DIV 32;
000024            Y1 := Y DIV 32;
000026
000026            (* DO HOUSEKEEPING *)
000026            PTR := DSSPTR^.TARGARRAY[X1,Y1].TARG;
000037            IF PTR <> NIL THEN BEGIN    (* COINCIDES WITH A REAL TARG *)
000041                IF (NOT PTR^.TOUCHED) AND (NOT DISABLED(ID,ID1)) THEN BEGIN
000054                    ID := DSSPTR^.TARGARRAY[X1,Y1].BLOCKID;
000066                    TARGID := PTR^.ID;
000074                    PTR^.TOUCHED := TRUE;
000100
000100                    (* DO VISUAL ACTIONS *)
000100                    CASE PTR^.STYLE OF
000105                        UNDERLINE : BEGIN
000105                            DRAWBOX(PTR^.X,PTR^.XLEN,PTR^.Y,PTR^.YLEN,DOTS,DOTS,LL,-2,ERASE,DSPNO);
000137                            SETCOORD(PTR^.LOBGX,PTR^.LOBGY,DOTS);
000152                            MODE(DSPTEXT,DSPNCNP,OVER,DSPNO);    (*SET IN MODE TO UNDERLINE *)
000157                            FOR I := 1 TO CNTCHARS(PTR^.LABL,PTR^.LBLEN) DO BEGIN
000177                                WRITE(OUTPUT,'_');
```

```
000204  3538                            END;
000211  3539                        END;
000212  3540
000212  3541                        PAT : BEGIN
000212  3542                            DRAWBOX(PTR^.X,PTR^.XLEN,PTR^.Y,PTR^.YLEN,DOTS,DOTS,LL,-5,DSPNORM,DSPNO);
000244  3543                        END;
000245  3544                    END;        (* OF CASE *)
000251  3545                END
000251  3546                ELSE BEGIN
000252  3547                    ERROR := MISTOUCH;
000254  3548                END;
000254  3549            END
000254  3550            ELSE BEGIN
000255  3551                IF INCLUDEBLOCKS AND (NOT DISABLED(ID,ID1)) THEN BEGIN
000263  3552                    ID := DSSPTR^.TARGARRAY[X1,Y1].BLOCKID;
000275  3553                    TARGID := 0;    (* INDICATES WHOLE BLOCK *)
000277  3554                END
000277  3555                ELSE BEGIN
000300  3556                    ERROR := MISTOUCH;
000302  3557                END;
000302  3558            END;
000302  3559        END;
000302  3560    END;                        (* OF GETTARGINP *)
000334  3561
000334  3562
000334  3563
000334  3564    (*------------- M A I N -------------*)
000334  3565    BEGIN
000334  3566        (*$I'GBLINIT'/'KBLIB'*)

------ BEGIN INCLUDED TEXT.

000334  3566 (* INITIALIZATION OF GLOBAL VARIABLES *)
000334  3566
000334  3566        TMSPTR := NIL;
000040  3566        PRSPTR := NIL;
000041  3566        RESPTR := NIL;
000041  3566        DSSPTR := NIL;
000042  3566        S1SPTR := NIL;
000043  3566        S2SPTR := NIL;
000043  3566        GBLERROR := NOERROR;

------ END INCLUDED TEXT.

000044  3567         INITDSPARRAYS('CHTEXT    ');
000046  3568        RESET(MSG) ;
000050  3569        CREATEBLOCK(DSSPTR^.I,0,512,0,512,DOTS,DOTS,LL,5,NOSCROLL,WRAP,STANDARD,ALTERNATE,GBLERROR);
000075  3570        DSTEXT(1,MSG,GBIERROR);
000100  3571        WITH DSSPTR^ DO BEGIN
000105  3572            WHILE TRUE DO BEGIN
000107  3573                GETTARGINP(I,J,CBUF,K,FALSE,GELERROR);
000121  3574            END;
000122  3575        END;
000122  3576    END.

        COMPILER ESTIMATED 'W' OPTION = 011555B.
```