DM80.WS4 (= Display Manager-80 version 1.0 article)

- "Display Manager from DRI"
  Alan Simpson
  "Microsystems", February 1984, p.96

(Retyped by Emmanuel ROCHE.)

Most professional programmers wince a bit when they hear the term "I/O code." Not because writing code for data entry screens and reports is difficult, but rather because it is a boring and tedious task. Typically, the programmer designs displays on graph paper, then laboriously writes line after line of code to format displays on the screen and printer. The process becomes even more unpleasant if the program is to be used with many different terminals. The programmer then needs to take into consideration the control codes for various CRTs. This is a very time-consuming process, particularly if one plans on making one's software compatible with 50 different terminals. Most professional programmers get around some of the tedium by writing general-purpose I/O routines, and storing displays and terminal codes on data files.

Digital Research has come up with an even better method. You, the programmer, buy Display Manager, then you "draw" your input and output displays directly on the screen, exactly as you wish them to appear at run-time. Display Manager then takes care of writing the I/O functions, storing displays on a data file, and providing control codes for a variety of terminals. Sounded good to me, so I thought I would give it a try.

Display Manager (DM-80) is one of Digital Research Incorporated's ("DRI") "Productivity Tools" (ROCHE> The other one is "Access Manager", providing ISAM (Indexed Sequential Access Method) to DRI's compiled languages.), and works with any of their 8-bit programming languages (CB-80, Pascal/MT+, or PL/1-80), and 16-bit languages, including Pascal/MT+86, CB-86, PL/1-86, and DRC. Version 1.0 of DM-80, the one I used for this review, supports 55 different terminals, and allows the programmer to include extra terminals. Display Manager also includes a program written in CB-80 (CBASIC Compiler Version 2) that allows the end user to install the program to his particular terminal. DM-80 requires that you use CP/M, CP/NET, or MP/M, and have at least 40 Kilobytes available in the Transient Program Area of your main memory. Display Manager will also run under PC DOS with any of the DRI 16-bit programming language.

Using Display Manager

When I first received DM-80, I read the manual from cover to cover. Like most software manuals, the DM-80 manual tends to be more descriptive than tutorial. The first thought that came to mind after reading the manual was "What?" The manual is about 100 pages in length, with the usual addenda that tell you what the manual forgot to mention, as well as changes that have been made since the printing of the manual (yes, even though this is Version 1.0). At first, I was dubious as to whether or not this product was truly going to help increase my

productivity. Since I have already developed a number of my own canned functions to handle I/O screens, it seemed unlikely that learning this new and seemingly complex tool would be worth the effort. When I actually sat down and used DM-80, I found it much easier to use than expected, and well worth learning.

Using DM-80 is essentially a 3-step process: 1) Install DM-80 to your particular terminal; 2) Create and edit displays using the DM-80 editor; and 3) Write the application programs that access the displays. I will discuss my experiences with each step in the process.

1. Installation

Installing DM-80 to your particular CRT is a simple process, unless you happen to be using 5 1/4" disks. DM-80 is delivered on two 8" disks, and one needs to do quite a bit of wading through the manual to determine exactly which files must be resident on disk during the various phases of designing screens. I managed to get DM-80 up and running on a single-sided double-density (180K) disk through a little trial and error. Once you have the correct files on disk, the rest is easy. DM-80 is menu-driven, and the program itself is somewhat tutorial.

If you are using one of the DM-80 supported terminals, installing the program is as simple as selecting that terminal from a menu of choices. Of the 55 terminals that DM-80 supports, many are different models from the same manufacturer. For purposes of brevity, I will just list the manufacturers here:

A.B.M.
A.D.D.S.
Apple
Beehive
Control Data
Cromemco
Digital Equipment Corporation (DEC)
Direct
Hazeltine
Heath
Hewlett-Packard
I.S.C.
Lear-Siegler
Microterm
Osborne
Radio Shack (Tandy)
Soroc
Teleray
Televideo
Toshiba
Vector Graphic
Visual Technology
Xerox
Zenith

If you are not using one of the supported terminals, you will have to provide the control codes for a custom terminal. This is not difficult, provided that the custom terminal has enough documentation to supply the appropriate codes. The DM-80 manual has a simple questionnaire to fill out about custom terminal characteristics. Then, the install program asks the same questions that the questionnaire did, and you fill in the blanks. The installation program has a very convenient test capability that allows you to check, to make sure you have installed a custom terminal properly. It does so by trying each function (clear screen, position cursor, reverse video, etc.) on the screen, and asking if the function worked correctly. If you discover a mistake during the test phase, you can edit the terminal codes using a reinstall option. Once you have DM-80 installed for your system, you can begin creating displays.

2. Creating displays

I never thought I would see the day I would actually enjoy creating I/O screens. DM-80 changed that, by allowing me to draw and edit displays on the screen in an interactive, visual manner.

When you call up Display Manager's editor, it asks if you want to edit an existing display, or create a new one. If you create a new one, it must have a unique number, as this number is used by the application program for finding the display. When you are ready to create your display, the editor presents a blank screen with the cursor in the upper left-hand corner, and you can just start drawing your display on the screen as you wish it to appear to the user at run-time.

The manual tends to make this process more difficult than it is. There are well over 40 distinct control-key commands (some 3 characters long!) that the editor uses. Personally, my brain's RAM space for storing control-key sequences is just about full, but DRI was quite considerate in making memorization a bit easier. For instance, many of the control-key sequences are identical to those used in other software packages (^V toggles insert mode, ^OC centers, while ^A, ^S, ^E, ^D, ^X, ^S, ^F move the cursor about on the screen, etc.). DRI also provides abbreviated reference cards, kindly laminated in clear plastic, for quick reference. When you first start designing displays, however, be prepared to do a good deal of wading through the manual. Control-key definitions are interspersed throughout the text, and the reference cards are too brief for first-time use.

When you are developing a display, you simply type out the prompts, headings, and borders where you want them to appear on the screen. You can also enter a control key command to specify that either an input or output field be displayed. You can easily move text and fields about the screen, as you zero in on just the format you wish. You can also include template characters in input fields, such as "(___) ___-____" for phone numbers. Then, you can determine visual characteristics for the various fields in a simple and pleasant manner. To do so, simply position the cursor at the beginning of a field, and enter a control key command to call up the status window. The following then appears on the screen.

```
+-------------------------------------------------+
| Field No.  Row  Col  Len     Posts     Type-OUTPUT |
|   000     000  000  000      YY       *rr,cc*nnn  |
+-------------------------------------------------+
| Validate  :X:  X,A,C,D,F,I,U    Beep     :N:   N,Y |
| Format    :L:  L,R,N,0-9,C,M    AutoRet  :N:   N,Y |
|                                   |
| End input---Cursor :N:   BadC :N:  FKey :N:    N,Y |
| Video  :N:  :N:  :N:  :N:  :N:  :N:  :N:  :N:  N,Y |
|       Invs Half Invr Flsh Undl Usr1 Usr2 Usr3     |
+-------------------------------------------------+
```

This window presents the DM-80 default characteristics for a single input field on the screen display. You can use the default characteristics for this field, or change them by simply moving the cursor about the status window. Of course, you can change default characteristics also.

The top line of the status window presents the field number (automatically assigned as the display is being designed), the row, column, and length of the field, whether or not it is surrounded by blank spaces (posts), and the type of field (INPUT or OUTPUT). The letters rr and cc are the row and column numbers of the cursor's present position on the screen, and nnn is the number of fields in the display.

The Validate prompt allows the programmer to provide error checking with the simple press of a key. The options are X (accepts any printable character), A (accepts only alpha characters), C (all characters, including control characters), D (decimal numbers only), F (allows Function keys only), I (integer only), and U (same as X, but input is changed to uppercase). Beep determines whether or not an illegal entry by the user causes the terminal's bell to ring (Y/N).

Format for the fields can be L (left-justify), R (right-justify), N (numeric output), 0-9 (number of digits to the left of the decimal point), C (send control keys to the screen), and M (money fields with leading dollar [or other currency] sign and 2 digits to the right of the decimal point).

The AutoRet option determines whether data entry terminates when the field's capacity is full (Y/N). The End-Input options allow the programmer to specify various methods for terminating data entry. If cursor is selected (Y), then the terminal's up/down arrow keys terminate data entry for the field. If BadC is Y, any illegal character for the field terminates entry for that field. FKey terminates entry if a Function key is entered.

The remaining options Invs, Half, Invr, Flsh, Undl, Usr1, Usr2, Usr3 allow the programmer to specify visual attributes for a field. By filling in a Y above an option, the programmer can cause the field to be invisible, half intensity, inverse video, underlined, or flashing. The programmer can also define up to 3 user-defined visual attributes, and include these in various fields.

The whole procedure is simple and fast. You just draw the display as you want it to appear to the end user at run-time, then set the cursor to the beginning of each input and output field, and use the status window to determine the basic characteristics and visual attributes of the individual field. Any

programmer who has ever written I/O code to include as many options as Display Manager provides will probably see that this is a far quicker and easier method. I was certainly convinced.

As if this were not enough, Digital Research took it a step further, and made Display Manager self-documenting. Once the display is created, the programmer can store a print-image ASCII file of the display on a disk file. This image file can be pulled directly into most word-processing systems, to ease the development of a user's manual. Also, the disk file contains detailed documentation for each field in the display, which helps with the technical documentation, as well as with debugging and modification. The final step in the process is to link your displays with your application program.

3. Write the application program

Once the displays have been designed, you need to write the actual programs that will use the displays. Display Manager adds the following functions to the programmer's present language:

INITDM
Initializes the application program to use a specific terminal's control codes and capabilities.

OPENDIS
Opens a DM-80 display file.

RETDM
Returns visual attributes supported by a given CRT, so the programmer knows which of DM-80's options are readily available for a particular CRT.

CLRSCR
Clear-the-screen command.

CURS
Set cursor to visible or invisible.

DISPD
Places a display from the display file onto the screen.

CLSDIS
Closes a display file.

For managing the actual fields in the display from the application program, DM-80 provides the following functions:

POSF
Positions the cursor to a given field.

NXTF
Positions the cursor to the next field.

SETF

Modifies the visual attributes of a field during run-time.

RETF
Returns the field position, length, and type.

PUTF
Outputs data to the current field.

GETF
Accepts and validates data entered to a field by the end user.

UPDF
Updates and validates data entered for a field.

ENDF
Determines how user ends data entry.

RESF
Resumes operation at last field.

The syntax for using most of DM-80's functions (using CB-80 as an example) is:

```
integer variable = FUNCTION (integer expression)
```

Since DM-80 uses functions rather than commands, it is the programmer's responsibility to determine whether or not a function is successful, and to return an error message describing the problem to the user, should an error occur. This adds a bit of bulk to an application program, but then again, it does provide the programmer with some flexibility in handling errors.

There are some minor annoying inconsistencies among the functions that the programmer must deal with. For example, some functions return a zero when the function is successful (Boolean false?), and negative value when the function is unsuccessful (Boolean true?). Some functions, like the CURS function, allow various numeric arguments (e.g., 0-3), but the value must be expressed as a string. Other functions do not use strings for numeric arguments. Until you get used to the exact syntax of the various functions, plan on doing a bit of debugging. You may find some of the syntax awkward and counter-intuitive at first.

The final step is to write the program in the language of your choice, and use the various DM-80 routines to access displays and manage field data. In your source program, you need to include DM-80's prewritten functions. For example, in CB-80, you need to include the command:

```
%INCLUDE dm80extr.bas
```

Digital Research provides external functions for all the supported languages. Then, when you link the compiled code, you need to include the DM-80 relocatable library as an overlay. In CB-80, the command to do so is:

```
A>LK80 testprog,dm80cb80.irl
```

Digital Research provides run-time libraries for each of the supported

languages. The manual provides sample programs written in CB-80, Pascal/MT+, and PL/1-80 as useful examples of programming techniques.

Incidentally, once the source code is written and is capable of putting displays on screen, the end-user can use control keys or arrow keys to move the cursor about on the screen. DM-80 defaults to both the ANSI standard keys for moving the cursor about (^H, ^J, ^K, ^L), as well as the more popular ^S, ^X, ^E, ^D keys. The programmer needs not write any code to provide these capabilities.

Similar products

To my knowledge, there are no products similar to Display Manager for compilable languages. Ashton-Tate's dBase II, however, includes a program called ZIP that provides a capability similar to, but not as flexible as, DM-80. Both ZIP and dBase II have one advantage over DM-80 and the DRI compilable languages: they are easier to use. With ZIP, the programmer draws the display on the screen, and follows prompts with the actual field or variable name that the prompt will be expecting. The programmer can also place commands on the screen that will later be embedded in the source code. ZIP then generates source code for the screen displays.

The programmer pays a heavy price for this ease of use, however, and here is where Display Manager shows its true advantages. First, DM-80 can be used with high-performance native-code compilers, whereas ZIP can only be used with dBase II, a slow-running interpretive language. For the independent software developer, DM-80 allows the user to write programs that will run on just about any 8- or 16-bit based systems, and Digital Research does not charge royalties to the developer. ZIP and dBase II narrow the market to customers who already own dBase II, unless the developer is willing to pay some rather astronomical "royalty" fees ($70-$100 per copy!) to Ashton-Tate for a run-time package that allows non-dBase II owners to use the package. Unfortunately, the dBase II run-time package slows the applications programs down even further. Also, dBase II is a very high-level database management system which, while providing powerful commands, robs the more sophisticated programmer of some lower-level flexibility, such as arrays, mathematical functions, and the ability to have more than 2 data files active at any time. Basically, if you are already an experienced programmer and you prefer a compilable language, DM-80 is your best bet. If not, perhaps ZIP and dBase II are preferable.

Recommendation

I found DM-80 to be a very powerful and productive programming tool. It is also a pleasure to work with, though somewhat awkward at first. I would recommend it highly to any professional programmer who is already fluent in any of the DRI compilable languages. I would especially recommend DM-80 to anyone thinking about writing marketable software, as it will greatly reduce the labor inherent in making your programs compatible with a variety of terminals.

Display Manager is available from Digital Research, and costs $400 for the 8-bit version, $500 for the 16-bit version. You can call Digital Research at (...) for a dealer or distributor nearest you.

EOF

DMCARD1.WS4 (= Display Manager Card #1)

- "Digital Research -- Display Manager Editor Commands"

(Retyped by Emmanuel ROCHE.)

Hold Ctrl key down for all commands

Cursor movement commands

D or L	Cursor right
S or H	Cursor left
E or K	Cursor up
X or J	Cursor down
RETURN key	Cursor next line

General editing commands

A	Left word
F	Right word
G	Delete character under cursor
T	Delete right word
V	Insert space
I	Tab
DEL key	Delete left character

Field-specific commands

UA	Cursor to previous field
UC	Copy field
UD or UL	Cursor to end of field
UF	Cursor to next field
UG	Delete field
UI	Define input field begin -- ESC ends
UM	Move field to cursor
UO	Define output field begin -- ESC ends
UR	Renumber fields
US or UH	Cursor to beginning of field
UV	Move field one space right, or Add space before cursor
UW	Set current status window as default
UZ	Delete field, turn initial values to literals

Miscellaneous commands

B Show field boundaries -- next key resets
P Set initial value character as template
W Change field attributes in status window -- ESC ends
OB Turn border move on/off, Cursor sets char -- ESC ends
OC Center line
ESC key End certain commands (No Ctrl needed)

Accelerated commands

QB Show boundaries of all fields -- ESC ends
QD or QL Screen right
QS or QH Screen left
QE or QK Screen top
QG Delete line
QV Insert line
QW Leave status window on -- Ctrl-QW ends
QX or QJ Screen bottom
QY Show visual attributes all fields -- next key ends

Output commands

OUN Save and edit next display
OUP Save and edit previous display
OUQ Abandon and exit to main menu
OUS Save display and continue editing
OUT Save display and exit to main menu
OUW Write documentation (file/printer)

Field status window defaults

Input field status window

```
+-------------------------------------------------+
| Field No.  Row   Col   Len   Posts  Type-INPUT   |
|  000     000   000   000    YY    *rr,cc* nn    |
+-------------------------------------------------+
| Validate   :X:   X,A,C,D,F,I,U  Beep    :N:  N,Y  |
| Format     :L:   L,R,N,0-9,C,M  AutoRet :N:  N,Y  |
|                                  |
| End Input -- Cursor :N:  BadC :N:  Fkey :N:  N,Y  |
|                                  |
|    :N:   :N:  :N:  :N:  :N:  :N:  :N:  :N:  N,Y  |
|   Invs  Half Invr Flsh Undl Usr1 Usr2 Usr3      |
+-------------------------------------------------+
```

Output field status window

```
+-------------------------------------------------+
| Field No.  Row   Col   Len   Posts  Type-OUTPOUT |
|  000      000  000   000    YY    *rr,cc* nn    |
+-------------------------------------------------+
| Format    :L:   L,R,N,0-9,C,M    Comma :N:  N,Y  |
|                                    |
|    :N:   :N:  :N:  :N:  :N:  :N:  :N:  :N:  N,Y  |
|   Invs  Half Invr Flsh Undl Usr1 Usr2 Usr3      |
+-------------------------------------------------+


EOF
```

DMCARD2.WS4 (= Display Manager Card #2)

- "Digital Research -- Display Manager Run-time Functions"

(Retyped by Emmanuel ROCHE.)

Run-time functions

```
I  = CLRSCR     Clear screen
I  = CLSDIS     Close current display file
C$ = CURS(D$)   Make cursor visible or invisible, return current setting
I  = DISPD(J)   Show display with reference number J from current display file
K  = ENDF       Return  0 if last input terminated normally; otherwise,  value
                of terminating character or negative Function key number.
C$ = GETF       Get input from field
I  = INITDM(D$) Initialize Run-time Library with CRT control code
I  = NXTF(J)    Move to next or previous field in screen order
I  = OPNDIS(D$) Open display file
I  = POSF(J)    Move to field with number J
I  = PUTF(D$)   Write D$ to current input or output field
C$ = RESF(J)    Resume  input  from field stored with call to -J.  Return  all
                data in field.
C$ = RETDM      Return attribute capabilities of CRT
C$ = RETF       Return field position, length and type
C$ = SETF(D$)   Set  or  reset  video  attributes  of  field;  return  current
                settings.
C$ = UPDF       Get data from field. If input field, get input to update data.
```

Notes:
- I, J and K are integers.
- C$ and D$ are character strings.
- All functions returning to I, return a negative number if error occurs.
- Pascal/MT+ string returning functions use the following convention:

```
char := STRINGFUNCT(PARAM,STRINGRET).
```

Specific function values

```
Function  Parameter                  Returned value
--------  -------------------------  ---------------------------
CURS      "0" = Reset to normal mode    "0" = Currently set to normal
SETF      "1" = Set to special mode     "1" = Currently set to special
          "2" = Switch to opposite
          "3" = No change
RETDM                                "0" = Not supported
                                     "1" = Supported as PAINT
                                     "2" = Supported as PLANT
```

NXTF FIELD TYPE Current field number; -1 if error

MOVE either input output

TO +----------

next | 1 2 3

prev | -1 -2 -3

last | 10 20 30

first | -10 -20 -30

RESF -J = Store current field number If J is negative, then RESF
cursor position and field data returns a null string.
(J = 1 to 8).
J = Write back data in field If J is positive, then RESF
at time of input termination; returns all data input using
resume input. the previous input function.

Attribute value positions

	Function		
Attribute	CURS	SETF	RETDM
---	---	---	---
Cursor on/off	1		1
Visibility		1	
Half intensity		2	2
Reverse video		3	3
Flashing		4	4
Underlining		5	5
User def. 1		6	6
User def. 2		7	7
User def. 3		8	8

Note: Values = string position

EOF

DMFS.WS4 (= Display Manager Function Summary)

- "Digital Research -- Display Manager Function Summary"

First Edition: September 1983

(Retyped by Emmanuel ROCHE.)

Table of Contents
---

1 Introduction

The "Display Manager Function Summary" contains information extracted and summarized from your other Display Manager manuals. Explanations of the commands and functions provided here are brief. If you require more detailed information, please consult your Display Manager manuals.

Display Manager commands are used in conjunction with the Editor program. Display Manager functions are routines contained in the run-time library.

The "Display Manager Function Summary" does not contain information specific to any operating system or programming language. Consult the appropriate programmer's guide for this information.

Note: Due to limitations of 8-bit microprocessors, color features are not available. If your computer is based on the Zilog Z-80 or Intel 8080 microprocessor, you should disregard any references to color.

2 Status windows

To display the status window for a field, place the cursor within the boundaries of the field and enter a Ctrl-W. The status window appears on the screen with the cursor positioned inside.

To display the status window constantly, regardless of cursor location, use the Ctrl-QW command. The cursor is not positioned inside the window.

Both Ctrl-W and Ctrl-QW are toggle commands; re-enter the command to remove the status window from your screen. Use the cursor movement commands, the RETURN key, or the space bar to move the cursor while inside the status window.

Status windows for input and output fields are similar, but distinctly and logically different.

```
+-----------------------------------------------------+
| Field No.     Row   Col   Len   Posts   Type-OUTPOUT |
|  nnn         nnn   nnn   nnn    YY     *rr,cc* nn   |
+-----------------------------------------------------+
| Format    :L:   L,R,N,0-9,C,M      Comma :N:   N,Y  |
|                                   |
| Video   :N:   :N:  :N:  :N:  :N:  :N:  :N:  :N:  N,Y  |
|       Invs  Half Invr Flsh Undl Usr1 Usr2 Usr3       |
| Color   :N:   :N:  :N:  :N:  :N:  :N:  :N:  :N:  N,Y  |
|        fls   RED  GRN  BLU  Int  red  grn  blu       |
+-----------------------------------------------------+
```

Figure 1. Output field status window

```
+-----------------------------------------------------+
| Field No.     Row   Col   Len   Posts   Type-INPOUT  |
|  nnn         nnn   nnn   nnn    YY     *rr,cc* nn   |
+-----------------------------------------------------+
| Validate  :X:   X,A,C,D,F,I,U     Beep   :N:   N,Y  |
| Format    :L:   L,R,N,0-9,C,M     AutoRet :N:   N,Y  |
|                                   |
| End Input -- Cursor   :N:   BadC :N:   Fkey :N:  N,Y  |
| Video   :N:   :N:  :N:  :N:  :N:  :N:  :N:  :N:  N,Y  |
```

```
|       Invs  Half Invr Flsh Undl Usr1 Usr2 Usr3       |
| Color   :N:   :N:  :N:  :N:  :N:  :N:  :N:  :N:  N,Y  |
|       fls   RED  GRN  BLU  Int  red  grn  blu       |
+-----------------------------------------------------+
```

Figure 2. Input field status window

2.1 Validation codes

Validate :X: X,A,C,D,F,I,U

The code you enter between the colons (":") specifies the type of validation you want performed on an input field.

Code	Meaning
X	Any printable character is accepted (X is the default).
A	Only alphabetic characters, including spaces, are accepted.
C	Any characters, including control characters, are accepted; function key input is not interpreted. Data entry for these fields can only be terminated using the RETURN key.
D	Only signed, decimal data is accepted.
F	Only function keys are accepted.
I	Only signed, integer data is accepted.
U	Same as type X, except that all information is converted to uppercase.

2.2 Format codes

Format :L: L,R,N,0-9,C,M

The format code you enter between the colons (":") specifies a particular output format for a field.

Code	Meaning
L	Left-justified.
R	Right-justified.
N	Numeric format. Numbers are right-justified, and leading zeros removed.
0-9	Decimal format. The number indicates how many positions to the right of the decimal point.

C Use for sending control characters to the screen. Information is displayed unformatted. See your reference manual for further information.

M Money format. Inserts a currency symbol, and aligns two digits to the right of the decimal point.

2.3 Video codes

```
Video  :N:  :N:  :N:  :N:  :N:  :N:  :N:  :N:  N,Y
     Invs  Half Invr Flsh Undl Usr1 Usr2 Usr3
```

If color is available on the run-time terminal, Display Manager uses the color codes in place of the video codes. The following descriptions explain the results of setting the video code in the field's status window to Y (the default is N). With the exception of Invs, all codes require that the feature selected must be available on the run-time terminal.

Code	Result
Invs	The initial value for the field does not appear when the display is shown on the run-time terminal.
Half	The field is displayed in half-intensity.
Invr	The field is displayed in inverse video.
Flsh	Characters in the field flash on and off.
Undl	Characters in the field are underlined.
Usr1-3	User-defined video attributes one, two, and three, respectively, are activated for the field.

2.4 Color codes

```
Color  :N:  :N:  :N:  :N:  :N:  :N:  :N:  :N:  N,Y
     fls  RED  GRN  BLU  Int  red  grn  blu
```

Set the "fls" color code to Y to cause the field to flash on and off, provided this feature is available on the run-time terminal.

The following tables show the colors that normally result when you see these color code combinations in the field's status window.

Background color codes

RED	GRN	BLU	Result

N	N	N	Black
N	N	Y	Blue
N	Y	N	Green
N	Y	Y	Cyan
Y	N	N	Red
Y	N	Y	Magenta
Y	Y	N	Brown
Y	Y	Y	White

Foreground color codes

Int	red	grn	blu	Result
N	N	N	N	Black
N	N	N	Y	Blue
N	N	Y	N	Green
N	N	Y	Y	Cyan
N	Y	N	N	Red
N	Y	N	Y	Magenta
N	Y	Y	N	Brown
N	Y	Y	Y	White
Y	N	N	N	Gray
Y	N	N	Y	Light blue
Y	N	Y	N	Light green
Y	N	Y	Y	Light cyan
Y	Y	N	N	Light red
Y	Y	N	Y	Light magenta
Y	Y	Y	N	Yellow
Y	Y	Y	Y	Bright white

2.5 Other status window codes

The following descriptions explain the results of setting the codes in the field's status window to Y. The default setting for these codes is N.

Code Result

Beep Sounds the terminal's audio beeper (if available) when the end-user enters unacceptable information into the field based on the field's validation code.

Comma Inserts a comma to the left of every third digit to the left of a decimal point in a numeric field.

AutoRet Automatically terminates data entry when the end-user fills the field with information.

Cursor Automatically terminates data entry when the end-user enters the up or down cursor movement or cursor arrow keys.

BadC Automatically terminates data entry when the end-user enters a

character not conforming to the field's validation code.

Fkey Automatically terminates data entry when the end-user enters one of the supported function keys.

3 Editor commands

Editor commands are listed here alphabetically according to their description. There are alternate commands to many of those listed. Please refer to your Display Manager manuals for more information.

Command description	Command
Abandon without saving display	Ctrl-OUQ
Beginning of field	Ctrl-US
Beginning of next line	RETURN key (Ctrl-M)
Boundary display (all fields)	Ctrl-QB
Boundary display (single line)	Ctrl-B
Center line	Ctrl-OC
Change field to literal	Ctrl-UZ
Change global values/save display	Ctrl-OUG
Copy field to cursor location	Ctrl-UC
Define input field	Ctrl-UI
Define output field	Ctrl-UO
Delete character to left	DEL key (Ctrl-H)
Delete character under cursor	Ctrl-G
Delete field	Ctrl-UG
Delete line	Ctrl-QG
Delete word to right	Ctrl-T
Down half screen	Ctrl-QX
Down one line	Ctrl-X
Draw border	Ctrl-OB
End of field	Ctrl-UD
Insert line	Ctrl-QV
Insert space	Ctrl-V
Left half screen	Ctrl-QS
Left one space	Ctrl-S
Move field right	Ctrl-UV
Move field to cursor location	Ctrl-UM
Next field	Ctrl-UF
Next word	Ctrl-F
Prepare documentation for display	Ctrl-OUW
Previous field	Ctrl-UA
Previous word	Ctrl-A
Renumber fields	Ctrl-UR
Right half screen	Ctrl-QD
Right one space	Ctrl-D
Save display, edit next one	Ctrl-OUN
Save display, edit previous one	Ctrl-OUP
Save display, edit same one	Ctrl-OUS
Save display, return to Main Menu	Ctrl-OUT

Set status window as default	Ctrl-UW
Status window display	Ctrl-W
Status window display (constant)	Ctrl-QW
Tab	TAB key (Ctrl-I)
Template insertion toggle	Ctrl-P
Up half screen	Ctrl-QE
Up one line	Ctrl-E
Video/Color attributes display (*)	Ctrl-Y
Video/Color attributes display	Ctrl-QY

(* = This command is inoperative in the 8-bit version of Display Manager.)

4 Summary of functions

This section summarizes the Display Manager functions. A syntax line, an explanation, and, when appropriate, additional information is provided for each function. The following is a list of the functions explained in this section.

Function description	Mnemonic
Clear screen	CLRSCR
Close display file	CLSDIS
Determine data entry termination method	ENDF
Determine field position, length, type	RETF
Display data in field	PUTF
Initialize run-time terminal and program	INITDM
Modify field attributes	SETF
Open display file	OPNDIS
Place cursor in relative field	NXTF
Place cursor in specific field	POSF
Place display on screen	DISPD
Resume data entry	RESF
Retrieve/validate user-entered field input	GETF
Retrieve/validate field input (with initial value)	UPDF
Return run-time terminal attributes	RETDM
Set cursor visible/invisible	CURS

CLRSCR

Syntax:

<integer variable> = CLRSCR

Explanation:

Clears the screen of the run-time terminal to blanks in all positions. Always returns zero.

CLSDIS

Syntax:

<integer variable> = CLSDIS

Explanation:

Closes the currently open display file. Returns zero if close is successful; otherwise, returns a negative value.

CURS

Syntax:

<string variable> = CURS (<string expression>)

Explanation:

Makes the cursor visible or invisible, provided the run-time terminal has this feature.

Argument values:

0 Set cursor to visible state
1 Set cursor to invisible state
2 Change current setting
3 Do not change current setting

Return values:

0 Cursor is visible
1 Cursor is invisible

DISPD

Syntax:

<integer variable> = DISPD (<integer expression>)

Explanation:

Places the display you specify on the screen of the run-time terminal.

Argument values:

Display reference number (1 to 250).

Return values:

Returns the display reference number if the display is located; otherwise, returns a negative value.

ENDF

Syntax:

<integer variable> = ENDF

Explanation:

Returns a value indicating how the end-user terminated data entry.

Return values:

0	(ASCII null) Normal termination.
x	Abnormal termination. x is the ASCII value of the invalid character causing termination.
-n	-n is a negative number indicating the function key that was pressed.

GETF

Syntax:

<string variable> = GETF

Explanation:

Returns information entered into the field. Initial field values are not returned.

INITDM

Syntax:

<integer variable> = INITDM (<string expression>)

Explanation:

Initializes the run-time application program and the run-time terminal.

Argument values:

Program attributes and terminal control codes for the run-time terminal must be passed in the following format:

<program attributes> / <terminal control codes>

Table 1. Program attributes string

Position	Attribute
1	Money symbol
2	Decimal delimiter
3	Alphabetic character set
4	Cursor movement keys
5	Clock set

The default value for these program attributes is A.

Return values:

Returns zero if initialization is successful. See your Display Manager manuals for other possible values.

NXTF

Syntax:

<integer variable> = NXTF (<integer expression>)

Explanation:

Locates the cursor at the beginning of a field you specify in the argument.

Argument values:

1	NEXT input or output field
2	NEXT input field
3	NEXT output field
10	LAST field in display
20	LAST input field in display
30	LAST output field in display
-1	PREVIOUS input or output field
-2	PREVIOUS input field
-3	PREVIOUS output field
-10	FIRST field in display
-20	FIRST input field in display
-30	FIRST output field in display

Return values:

Returns the field reference number if the field is located; otherwise, returns a negative value.

OPNDIS

Syntax:

<integer variable> = OPNDIS (<string expression>)

Explanation:

Opens a display file you specify. If there is a display file currently open, it is closed before the new display file is opened.

Argument values:

Name of display file to be opened.

Return values:

Returns zero if the file is opened successfully; otherwise, returns a negative value.

POSF

Syntax:

<integer variable> = POSF (<integer expression>)

Explanation:

Places the cursor in a field you specify.

Argument values:

Field reference number of the field to receive the cursor. If zero is passed, the field reference number of the current field is returned.

Return values:

If the field is located or zero is passed as the function argument, the current field reference number is returned. If zero is passed as the function argument but no field is current, zero is returned. If a specified field cannot be located, a run-time error results.

PUTF

Syntax:

<integer variable> = PUTF (<string expression>)

Explanation:

Displays data in the current field.

Argument values:

Information to display in field. Must not exceed 132 characters.

Return values:

Returns zero if the function is successful; otherwise, returns a negative value.

RESF

Syntax:

<string variable> = RESF (<integer expression>)

Explanation:

Provides a way for your application program to resume data entry in a field following abnormal termination. This requires that the RESF function be called twice. Initially, it is called with a negative argument value that causes Display Manager to 'remember' the cursor location and the function (GETF or UPDF) in use at the time. RESF can then be subsequently called with a corresponding positive value to restore the cursor to its original location and resume data entry. Note that data entry does not resume with the original GETF or UPDF function call issued by your program.

Argument values:

A value (negative or positive) ranging from 1 to 8.

Return values:

Returns the same value as that returned by the original GETF or UPDF function used to retrieve data from the field.

RETDM

Syntax:

<string variable> = RETDM

Explanation:

Returns values indicating what features are available on the run-time terminal and the version of the Display Manager run-time library in use.

Return values:

Returns a 16-character string; only the first 11 characters are used.

1 Cursor visibility
2 Half intensity
3 Inverse video
4 Flashing field
5 Underline
6 User-attribute #1
7 User-attribute #2
8 User-attribute #3
9 Function keys
10 Run-time version number
11 Color/Monochrome indicator

Position 1 returns a "1" if the invisible cursor feature is available; otherwise, "0" is returned.

Positions 2 through 8 interpret as follows:

0 Feature is not available.
1 Feature available; requires Paint method.
2 Feature available; requires Plant method.

Position 9 indicates the number of function keys available.

Position 10 indicates the version of the run-time library currently in use.

Position 11 returns "0" if the run-time terminal is monochrome; "1" if the run-time terminal is color-equipped.

RETF

Syntax:

<string variable> = RETF

Explanation:

Returns values indicating the position, length, and type (input or output) of the current field.

Return values:

Returns a 16-character string; only positions 1, 3, 5, 7, and 8 are

used.

1 Row number of the current field.
3 Column number where the current field begins.
5 Length of the current field.
7 "0" in this position means there is not a space on both sides of the field. "1" means there is a space on both sides of the field.
8 "I" in this position means the current field is an Input field; alpha "O" means the field is an Output field.

SETF

Syntax:

<string expression> = SETF (<string expression>)

Explanation:

Sets video or color attributes on the run-time terminal.

Argument values:

A 16-character string referring to specific attributes on the run-time terminal that can be set for the field. The following table shows the meaning of each character in the string with its normal (default) and special setting.

Table 2. SETF argument values

	Position	Attribute	Nrm	Spc
Video:	1	Invisibility	N	Y
	2	Half intensity	N	Y
	3	Reverse video	N	Y
	4	Flashing field	N	Y
	5	Underlining	N	Y
	6	User-defined attribute #1	N	Y
	7	User-defined attribute #2	N	Y
	8	User-defined attribute #3	N	Y
Color:	9	Flashing	N	Y
Background color codes:				
	10	RED	N	Y
	11	GREEN	N	Y
	12	BLUE	N	Y
Foreground color codes:				
	13	Intensity	N	Y
	14	Red	Y	Y
	15	Green	Y	Y
	16	Blue	Y	Y

The value of each position in the string indicates how you want the attribute set, as follows:

0 Normal setting.
1 Special setting.
2 Change current setting (normal to special, or vice versa).
3 Do not change current setting.

If the argument contains less than 16 characters, those for which no code is sent default to '3'.

Return values:

Returns a 16-character string indicating how attributes are currently set. The characters correspond to the attributes shown in the preceding table. If the corresponding position contains "0", the attribute is in the normal state; if it contains "1", it is in the special state.

UPDF

Syntax:

<string variable> = UPDF

Explanation:

Returns the character string entered into the field or its initial value.

5 Run-time errors

Run-time error codes contain two characters. The first identifies the function called when the error occurred; the second indicates the nature of the error. The following list shows the value and the corresponding function of the first error code character.

Letter	Function
a	CLRSCR
b	CLSDIS
c	CURS
d	DISPD
e	ENDF
f	GETF
g	INITDM
h	NXTF
i	OPNDIS
j	POSF
k	PUTF
l	RESF
m	RETDM

n RETF
o SETF
p UPDF

The following lists the value of the second error code character and a description of the error.

Value	Meaning
1	Function called prior to calling INITDM.
2	No display file currently open.
3	There is no current display on the terminal.
4	There is no current field in the display on the screen.
5	Second attempt made to use INITDM function.
6	RESF argument value is not between 1 and 8.
7	RESF function not previously called with negative argument value.
8	Not enough memory to show this display.
9	Target field of POSF function non-existent.
10	Wrong version of display file.
11	Not a valid display file.

EOF

DMPG80.WS4 (= Display Manager Programmer's Guide)

- "Display Manager Programmer's Guide"
  for the CP/M Family of Operating Systems

First Edition: July 1983

(Retyped by Emmanuel ROCHE.)

--> To be found... <--

DMPG86.WS4 (= Display Manager Programmer's Guide)

- "Display Manager Programmer's Guide"
  for the CP/M-86 Family of Operating Systems

First Edition: July 1983

(Retyped by Emmanuel ROCHE.)

Foreword

The "Display Manager Programmer's Guide for the CP/M-86 Family of Operating Systems" explains how to use Display Manager with Digital Research programming languages.

This "Programmer's Guide" is designed as a supplement to the "Display Manager Reference Manual". You need both books to make full use of Display Manager.

Section 1 describes general considerations for installing Display Manager. This section includes two tables that list and describe files on your Display Manager distribution disks.

The remaining sections explain how to use Display Manager with application programs written in one of the Digital Research programming languages supported by Display Manager.

Tables of Contents

Section 1: Installation guidelines

This section explains how to install Display Manager with an operating system in the CP/M-86 family of Operating Systems. You must complete the steps described below before you can use this productivity tool.

Getting started

1. Make a copy of your Display Manager distribution disks. Store the original disks in a safe place, and use the copy for all future processing.

2. Read the licensing agreement that comes with Display Manager. Complete the warranty/registration card, and return it to Digital Research. This registers you with our Customer Service and Technical Support departments. Then, you will be sure to receive news of changes made in the product.

3. Study your Display Manager documentation to become familiar with its contents and organization.

4. Display the directories of the disk copies made in Step 1. Your disks must contain the files listed in the following table. In the table, files are listed alphabetically by filename. This makes it easy to match them against your disk. If any Display Manager files are missing, contact Digital Research immediately.

5. Study the two tables in this section to determine which files you need for your particular situation. Then, configure one or more disks (preferably new disks containing an operating system) with the Display Manager files you require. For example, you might have one disk containing the terminal setup program and associated files, another containing the Editor program, and yet another containing the Run-time Library modules.

6. Create the Editor program for your design terminal by running the DMSET program. When you have completed this step, you are ready to

begin using the Editor to design displays.

Display Manager distribution files

The following tables list the files that should be on your distribution disks, with an explanation of what each file contains.

Table 1-1. Display Manager required files

Format: Filename
File description

DMCB.L86
The Run-time Library containing Display Manager functions used in CBASIC Compiler (CB-86) source programs.

DMDRC.L86
The Run-time Library containing Display Manager functions used in small or compact storage model C language source programs.

DMDRCBIG.L86
The Run-time Library containing Display Manager functions used in medium and big storage model C language source programs.

DMEDHLP.OVR and DMEDOVR.OVR
Program overlays used with the Editor. Use DMEDHLP.OVR for extended help. Use DMEDOVR.OVR for limited help.

DMEDU.CMD
The original version of the Editor program. Note that this version has not been created for use with any design terminal. See "Option E--Create Editor for Design Terminal" in Section 3 of your "Display Manager Reference Manual".

DMEXTR.BAS
Contains external declarations of Display Manager functions used with CBASIC Compiler (CB-86) source programs.

DMEXTR.C
Contains external declarations of Display Manager functions used with Digital Research C source programs.

DMEXTR.PAS
Contains external declarations of Display Manager functions used with Pascal/MT+86 source programs.

DMEXTR.PLI
Contains external declarations of Display Manager functions used with PL/I-86 source programs.

DMPASC.ERL
The Run-time Library containing Display Manager functions used in Pascal/MT+86 source programs.

DMPLI.L86
The Run-time Library containing Display Manager functions used in PL/I-86 source programs.

DMSET.CMD
Terminal setup program. This file is used to create the Editor for the design terminal, create a file of terminal control codes, change terminal control codes in the file, and other functions. DMSET.CMD is fully described in Section 3 of your "Display Manager Reference Manual".

DMSET.OVR, DMSET1.OVR, DMSET2.OVR, DMSET3.OVR, DMSET4.OVR, and DMSET5.OVR
Various program overlays used by the terminal setup program. These files must be on the same disk as the DMSET program.

TERMS.DM
A file containing the terminal control codes for terminals used with Display Manager. Appendix A of your "Display Manager Reference Manual" describes this file and the codes contained therein. This file must be on the same disk as the DMSET program.

Your distribution disks also contain other files that are not critical to the operation of Display Manager. These include sample programs and aids. These files are listed and explained in the following table.

Table 1-2. Other Display Manager files

Format: Filename
    File description

INSTALL.BAS
The source code for a program written in CBASIC Compiler. This program is designed for use in the run-time environment, to help the end-user set up the run-time terminal. The program uses the terminal control codes in the TERMS.DM file. You can make whatever changes you want to this program, and distribute it with your complete application programs.

ORDERS.DIS
A Display Manager display file containing displays used by the sample programs described later in this table.

PARTS.LST
A data file also used by the sample programs.

READ.ME
If present, this file contains information supplemental to your Display Manager documentation. Read this file before using Display Manager.

SAMDRC.C
A sample program, the same as SAMPLE.BAS, but written in Digital Research C.

SAMPAS.PAS
A sample program, the same as SAMPLE.BAS, but written in Pascal/MT+86. When

compiling this program, use the $B toggle, and link with BCDREALS.ERL and FULLHEAP.ERL.

SAMPLE.BAS
The source code for a sample program coded in CBASIC Compiler. You can examine, modify, and use this program in any way you want.

SAMPLE.CMD
A compiled form of SAMPLE.BAS. You can run this program by following these two steps:

1. Use the DMSET program to write the terminal control codes for your terminal into a file named CURRENT.TRM.

2. Type SAMPLE and press RETURN at your Operating System prompt.

SAMPLI.PLI
A sample program, the same as SAMPLE.BAS, but written in PL/I.

Section 2: CBASIC Compiler (CB-86) user's guide

This section explains how to use Display Manager with application programs written in CBASIC Compiler.

Linking CBASIC Compiler programs

To link a CBASIC Compiler program to the Display Manager Run-time Library, use the following command format in response to your Operating System prompt:

```
LINK86 <program name>,DMCB.L86
```

where <program name> is replaced by the name of the object module produced by CBASIC Compiler. For example, to link a program named MYPROG, use the following command line:

```
LINK86 MYPROG,DMCB.L86
```

Because DMCB.L86 is an indexed, relocatable library, the linker places it in the root module, in the event that language overlay structures are used.

CBASIC Compiler external declarations

CBASIC Compiler requires that you explicitly declare external functions. External functions are those not coded in the source program, but referenced by it. The file DMEXTR.BAS contains external declarations for all Display Manager functions. Use the %INCLUDE feature of CBASIC Compiler to make these external declarations a part of your application program.

Note: The %INCLUDE statement must precede any calls to Display Manager functions in your application program source code. Refer to the program listing at the end of this section for an example.

Function arguments and return values

Numeric values used as function arguments, or returned to your application program, must be of type integer. You can declare them with the INTEGER statement, or follow the name with a percent sign, %.

Character values used as function arguments, or returned to your application program, must be of type string. You can declare them with the STRING statement, or follow the name with a dollar sign, $.

The following example illustrates how you should declare function arguments and return values.

```
INTEGER Int.Value
STRING  Str.Value
...
INIT% = INITDM (Str.Value)      REM INIT% is an integer
...
More.In$ = RESF (Int.Value)     REM MORE.IN$ is a string
...
```

Sample program (SAMPLE.BAS) listing

The following listing shows the source code for program SAMPLE.BAS, which is provided on your distribution disks. The program is written in CBASIC Compiler language; you can modify or use it in any way you want. This listing is for reference only. Always treat the program on your distribution disks as the definitive version.

Listing 2-1. SAMPLE.BAS source code

```
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM     DISPLAY MANAGER -- CBASIC COMPILER SAMPLE PROGRAM
REM
REM                  JUNE 1, 1983
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM    All data entered with GETF and UPDF is in string form
REM
REM ------------------------------------------------------------------------
   STRING          \
     CUSTOMER,      \ Customer name
```

```
      ADDRESS,       \ Address
      CITY,          \ City
      STATE,         \ State
      ZIP,           \ Numeric value
      PHONE,         \ Numeric value
      PAYMENT(1),    \ Method of payment, and account number
      QTY(1),        \ Quantity of each item
      DESCRIPTION(1), \ Item description
      PART.NO(1),    \ 5 numeric digits (checked for validity)
      PRICE.EA(1),   \ Normal price inserted, but may be sale price
      TOTAL(1)       \ QTY * PRICE.EA

REM ------------------------------------------------------------------------
REM    Constants and arrays used by the program...
REM ------------------------------------------------------------------------
   ON$ = "0"                          REM Make a field visible
   OFF$ = "1"                         REM Make a field invisible
   LST.SZ% = 50                         REM Size of parts list
   TABS$ = "            "             REM Tabs for output

   DIM \
     PAYMENT(1),    \ Account number is second value
     QTY(4),        \ Only 5 different items allowed on one order
     DESCRIPTION(4), \
     PART.NO(4),    \
     PRICE.EA(4),   \
     TOTAL(4),      \
     PART.LST$(LST.SZ%,1),\
     PRICE$(LST.SZ%) \

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM    Include the Display Manager runtime library definitions
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
   %INCLUDE DMEXTR.BAS

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM    Get terminal control code from "CURRENT.TRM" file
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
   IF END # 1 THEN ERR1             REM If no file present, abort program
   OPEN "CURRENT.TRM" AS 1
   READ # 1;TERM$                   REM Read the terminal control code
   CLOSE 1                       REM Close the file

REM ------------------------------------------------------------------------
REM    Set up the Part Number List for the Help screen
REM ------------------------------------------------------------------------
   IF END # 2 THEN ERR2             REM If no file present, abort program
   OPEN "PARTS.LST" AS 2
   IF END # 2 THEN S.LS             REM Test for end of file
   FOR CNT% = 0 TO LST.SZ% - 1       REM Loop to build list of part numbers
```

```
      READ # 2; PART.LST$(CNT%,0),PART.LST$(CNT%,1),PRICE$(CNT%)
    NEXT
S.LS:PART.LST$(CNT% + 1,0) = ""        REM When end of list reached,
    CLOSE 2                    REM   close the file.

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM     Assign display reference numbers to displays
REM
REM     (these can be changed as neededby your program)
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
    PHONE.ORDER% = 1              REM Order Form display
    HELP% = 2                  REM Part Number reference display

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM     ERROR MESSAGES FOR FATAL DISPLAY MANAGER ERRORS
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
       INIDM$ = "ERROR: Initialization failure"
       OPNIS$ = "ERROR: Display file not found"
       DISD$ = "ERROR: Display not found"
       POS$ = "ERROR: Field missing"
       NXT$ = "ERROR: Next field missing"
       PUT$ = "ERROR: Write to field failure"
       CUR$ = "ERROR: Cursor On/Off failure"
       CLSDIS$ = "ERROR: Can't close display file"

REM ************************************************************************
REM
REM     ALL NON-DISPLAY MANAGER FUNCTIONS ARE DEFINED HERE...
REM
REM ************************************************************************

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM     This routine checks the value returned by any Display Manager
REM     function. Most functions return -1 if an error occurs. These
REM     are fatal errors, so the program is aborted.
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
    DEF DM.ERR(F.RET%,ERR.TYPE$)
      IF F.RET% >= 0 THEN RETURN          REM Not an error
      PRINT : PRINT                  REM Clear some space for err
      PRINT ERR.TYPE$                   REM Output message
      STOP                          REM Fatal, so quit
    FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM     This routine checks to see if the entered part number exists
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
```

```
   DEF SEARCH(PART.NO$)
     INTEGER SEARCH                    REM Return array position
     FOR CNT% = 0 TO LST.SZ% - 1
          IF PART.LST$(CNT%,0) = PART.NO$ \
               THEN SEARCH = CNT% : \
                    RETURN
          IF PART.LST$(CNT%,0) = "" \
               THEN GOTO ELST
     NEXT
ELST:  SEARCH = -1
   FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM    Data entry routine
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
   DEF GET.ENTRY
     STRING GET.ENTRY
     RET.ERR% = NXTF(2)                REM Move to next input field
     CALL DM.ERR(RET.ERR%,NXTF$)       REM Check for error in func
     ATTR$ = SETF(PRM.ON$)             REM Turn on the prompt
     INP$ = GETF                       REM Get field input
CONT:  IF ENDF = 27 \                  REM ESC key to exit?
          THEN CALL SETF ("000000") : \   REM Yes-Reset attributes
               CALL CLRSCR : \          REM Clear the screen
               RET.ERR% = CLSDIS : \    REM Close display file
               CALL DM.ERR(RET.ERR%,CLSDIS$) : \
               STOP
     IF ENDF <> 0 AND ENDF <> 26 \     REM Normal end, or ^Z entered?
          THEN GOTO RETR               REM No-ignore char and cont...
     GET.ENTRY = INP$                  REM Yes-Store the input
     ATTR$ = SETF(PRM.OFF$)            REM turn off the prompt
     RETURN                            REM Go back
RETR:  RET$ = RESF(-1)                 REM Save curr. field position
REM    RET.ERR% = PUTF(INP$ + PROMPT$)   REM Replace data in field
REM    CALL DM.ERR(RET.ERR%,PUTF$)       REM Check for error...
     INP$ = RESF(1)                    REM Resume input in org field
     GOTO CONT                         REM Continue data entry...
   FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM    Display error or help message
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
   DEF ERR.MSG(POS%,ONOFF$)
     RET% = POSF(0)                    REM Store current field numb.
     RET.ERR% = POSF(POS%)             REM Move curs. to spec. field
     CALL DM.ERR(RET.ERR%,POSF$)       REM Check for error
     ATTR$ = SETF(ONOFF$)              REM Turn message OFF or ON
     RET.ERR% = POSF(RET%)             REM Return to original pos.
     CALL DM.ERR(RET.ERR%,POSF$)       REM Check for error
   FEND
```

```
REM ************************************************************************
REM
REM     PROGRAM MAINLINE BEGINS HERE...
REM
REM ************************************************************************
    RET.ERR% = INITDM(TERM$)                REM Init Terminal and Library
    CALL DM.ERR(RET.ERR%,INITDM$)           REM Check for error
    AVAIL.ATTR$ = RETDM                     REM Get terminal attributes
    IF MID$(AVAIL.ATTR$,3,1) <> "0" \       REM If inv. video available,
      THEN PRM.ON$ = "031" : \
           PRM.OFF$ = "330" : \             REM   use for prompts.
      ELSE PRM.ON$ = "0" : \                REM Otherwise,
           PRM.OFF$ = "3"                   REM   just initials.

REM ------------------------------------------------------------------------
REM     Open display file, show it, and position to the first field
REM ------------------------------------------------------------------------
    RET.ERR% = OPNDIS("ORDERS.DIS")         REM Open display file
    CALL DM.ERR(RET.ERR%,OPNDIS$)           REM Check for error
LOOP:RET.ERR% = DISPD(PHONE.ORDER%)         REM Show Phone Order display
    CALL DM.ERR(RET.ERR%,DISPD$)            REM Check for error
    RET.ERR% = NXTF(-10)                    REM Put cursor in first field
    CALL DM.ERR(RET.ERR%,NXTF$)             REM Error check
    PROMPT$ = "_____"                       REM Initials

REM ------------------------------------------------------------------------
REM     If inverse video available, use for all prompts; otherwise, underline
REM ------------------------------------------------------------------------
    CUSTOMER = GET.ENTRY                    REM Use the Data Entry routine
    ADDRESS = GET.ENTRY                     REM   to enter data for these
    CITY = GET.ENTRY                        REM   fields...
    STATE = GET.ENTRY                       REM
    ZIP = GET.ENTRY                         REM
    PHONE = GET.ENTRY                       REM
PAYM:PAYMENT(0) = GET.ENTRY                 REM Payment code must be A, B,
    IF MATCH(PAYMENT(0),"ABC",1) = 0 OR \   REM   or C only.
      PAYMENT(0) = "" \                     REM If not A, B, or C,
      THEN CALL ERR.MSG(100,ON$) : \        REM   display error message,
           RET.ERR% = NXTF(-2) : \          REM   cursor to previous
           CALL DM.ERR(RET.ERR%,NXTF$) : \  REM   input field,
           GOTO PAYM \                      REM   and retry.
      ELSE RET.ERR% = NXTF(3) : \           REM Code o.k., next outp field
           CALL DM.ERR(RET.ERR%,NXTF$)      REM Check for error
    IF PAYMENT(0) = "A" \                   REM If payment on Account,
      THEN RET.ERR% = PUTF("CCOUNT") : \    REM   complete the word,
           CALL DM.ERR(RET.ERR%,PUTF$) : \  REM   check for errors,
           PAYMENT(1) = GET.ENTRY \         REM   and get account number.
    ELSE IF PAYMENT(0) = "B" \              REM If payment by bank card,
      THEN RET.ERR% = PUTF("ANK CARD") : \  REM   complete the word,
           CALL DM.ERR(RET.ERR%,PUTF$) : \  REM   check for errors,
           PAYMENT(1) = GET.ENTRY \         REM   and get account number.
    ELSE IF PAYMENT(0) = "C" \              REM If Cash On Delivery,
      THEN RET.ERR% = PUTF(".O.D.") : \     REM   complete the word,
```

```
        CALL DM.ERR(RET.ERR%,PUTF$)       REM   check for errors.
    CALL ERR.MSG(100,OFF$)                REM Turn message off

REM ---------------------------------------------------------------------
REM     Get ready to take the order...
REM ---------------------------------------------------------------------
    RET.ERR% = POSF(75)                   REM Move cursor to msg field
    CALL DM.ERR(RET.ERR%,POSF$)           REM Check for error
    ATTR$ = SETF("0")                     REM Make the field visible
    PROMPT$ = "             "             REM Initial characters
    ORDER.NO% = 0                         REM Init to first of 5 possi-
                                  REM   ble order entry items.
ORDR:QTY(ORDER.NO%) = GET.ENTRY           REM Get quantity for this item
    IF QTY(ORDER.NO%) = "0" THEN GOTO TTLS    REM If 0, order is complete
    DESCRIPTION(ORDER.NO%) = GET.ENTRY        REM Get item description
    CALL ERR.MSG(76,ON$)                  REM Turn on ^Z reference msg
PART:PART.NO(ORDER.NO%) = GET.ENTRY           REM Get part number (PN)
HRET:IF ENDF = 26 \                       REM If ^Z entered,
      THEN GOTO HELP                      REM   show parts list display.
    PART% = SEARCH(PART.NO(ORDER.NO%))        REM Otherwise, find PN in list
    IF PART% = -1 \ not valid part number     REM If part number not found,
      THEN CALL ERR.MSG(101,ON$) : \          REM   show the error message,
        RET.ERR% = NXTF(-2) : \           REM   replace cursor in field,
        CALL DM.ERR(RET.ERR%,NXTF$) : \   REM   check for error,
        GOTO PART                         REM   and try again...
    CALL ERR.MSG(101,OFF$)                REM Turn error message off
    CALL ERR.MSG(76,OFF$)                 REM Turn ^Z message off
    RET.ERR% = NXTF(2)                    REM Move cursor to Price field
    CALL DM.ERR(RET.ERR%,NXTF$)           REM
    RET.ERR% = PUTF(PRICE$(PART%))        REM Display the price
    CALL DM.ERR(RET.ERR%,PUTF$)           REM
    ATTR$ = SETF(PRM.ON$)                 REM Turn on the prompt
PTRY:PRICE.EA(ORDER.NO%) = UPDF           REM if CR, get initial value
REM ---------------------------------------------------------------------
REM     The price field does not trap bad characters or the ESC key
REM ---------------------------------------------------------------------
    ATTR$ = SETF(PRM.OFF$)                REM Turn off the prompt
    RET.ERR% = NXTF(3)                    REM Move cursor to total field
    CALL DM.ERR(RET.ERR%,NXTF$)           REM Check for errors
    TOTAL(ORDER.NO%) = STR$(VAL(QTY(ORDER.NO%)) * \
            VAL(PRICE.EA(ORDER.NO%)))   REM Compute total for item
    RET.ERR% = PUTF(TOTAL(ORDER.NO%))     REM Show the item total
    CALL DM.ERR(RET.ERR%,PUTF$)           REM
    ORDER.NO% = ORDER.NO% + 1             REM Get ready for next item
    IF ORDER.NO% < 5 THEN GOTO ORDR       REM Up to 5 items accepted
REM ---------------------------------------------------------------------
REM     Compute the total sale amount
REM ---------------------------------------------------------------------
    ORDER.NO% = 4                         REM Initialize loop control
                                  REM (0 to 4 for 5 items)
TTLS:SALE = 0                             REM Init the sale amount
    FOR CNT% = 0 TO ORDER.NO%             REM
      SALE = SALE + VAL(TOTAL(CNT%))      REM Add each item total
    NEXT                          REM
```

```
    RET.ERR% = POSF(26)                 REM Put cursor in Total Sale
    CALL DM.ERR(RET.ERR%,POSF$)           REM Check for error
    RET.ERR% = PUTF(STR$(SALE))         REM Write Total Sale amount
    CALL DM.ERR(RET.ERR%,PUTF$)           REM Check for error
    RET.ERR% = NXTF(20)                 REM Move to last input field
    CALL DM.ERR(RET.ERR%,NXTF$)           REM
    ATTR$ = SETF("0")                   REM Make the field visible
    RET$ = GETF                         REM wait for CR to be entered
    IF ENDF = 27 \                      REM If ESC entered,
      THEN GOTO DONE                    REM   that's all, folks!

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM     This program does not output the order information to a storage file.
REM     Insert your output routine(s) here to create a file with the order
REM     information...
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
    GOTO LOOP                           REM Take the next order

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM The following subroutine restores the original form to the screen
REM after the part numbers help screen has been shown. (See "HELP" below.)
REM This routine restores all information to the screen that was previously
REM entered by the end-user.
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
    DEF WRITEF(OUT$)
      RET.ERR% = NXTF(2)                REM Move to next input field
      CALL DM.ERR(RET.ERR%,NXTF$)         REM Check for error
      ATTR$ =  SETF("0")                REM Make the field visible
      RET.ERR% = PUTF(OUT$)               REM Show old data in field
      CALL DM.ERR(RET.ERR%,PUTF$)         REM Check for error
      RET$ = RETF                       REM Get current field specs
      IF POSF(0) = 8   \                REM If cursor is in Payment
            THEN RET.ERR% = NXTF(3) : \     REM   field, move to next
               CALL DM.ERR(RET.ERR%,NXTF$) : \  REM output field.
               RET.ERR% = PUTF(MID$(OUT$,2,LEN(OUT$))) : \  REM Display
               CALL DM.ERR(RET.ERR%,PUTF$)
    FEND

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM The following routine show the part numbers from the PARTS.LST file
REM when the end-user enters ^Z on the order form. This routine also
REM restores the original order form to the screen following the display
REM of the part numbers list. Also, note the use of the WRITEF routine
REM immediately above in restoring the original display.
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
HELP:RET$ = RESF(-1)                    REM Save current field no.
    CALL CURS("1")                      REM Make cursor invisible
    RET.ERR% = DISPD(HELP%)               REM Put HELP display on screen
```

```
    CALL DM.ERR(RET.ERR%,DISPD$)            REM Check for error
FIRS:CNT% = 0                           REM Initialize counter
    PAGE% = 0                          REM Init screen line count
NXTL:RET.ERR% = POSF(CNT% + 1)              REM Place cursor in field
    CALL DM.ERR(RET.ERR%,POSF$)            REM Check for error
    IF PART.LST$(CNT% + PAGE%,0) <> "" \      REM Display all items in list
      THEN TEMP$ = PART.LST$(CNT% + PAGE%,0) + TABS$ + \
              PART.LST$(CNT% + PAGE%,1) : \
          RET.ERR% = PUTF(TEMP$) : \
          CALL DM.ERR(RET.ERR%,PUTF$) : \
          CALL SETF("0") \
      ELSE CNT% = -1
    CNT% = CNT% + 1                    REM Bump the counter
    IF CNT% <> 0 AND CNT% < 22             REM End-list or screen full?
      THEN GOTO NXTL                    REM No-display next line
    RET.ERR% = POSF(100)                 REM Yes-cursor to end message
    CALL DM.ERR(RET.ERR%,POSF$)            REM Check for error
HRTR:RET$ = GETF                         REM Get end-user's response
    IF RET$ = CHR$(27) THEN GOTO REDS        REM ESC, redisplay order form
    IF RET$ <> CHR$(26) \                 REM If not ^Z,
      THEN GOTO HRTR                     REM   retry input response.
    IF CNT% = 0 THEN GOTO FIRS             REM If end of list, start over
    PAGE% = PAGE% + 21                   REM Otherwise, next page
    CNT% = 0                           REM Re-initialize counter
    GOTO NXTL                           REM Display the lines
REM ---------------------------------------------------------------------
REDS:RET.ERR% = DISPD(PHONE.ORDER%)          REM Bring back original disp
    CALL DM.ERR(RET.ERR%,DISPD$)            REM
    RET.ERR% = NXTF(-10)                 REM Put cursor in first field
    CALL DM.ERR(RET.ERR%,NXTF$)             REM   of display.
    CALL WRITEF(CUSTOMER)                REM WRITEF replaces original
    CALL WRITEF(ADDRESS)                 REM   data in each field.
    CALL WRITEF(CITY)                   REM   ...
    CALL WRITEF(STATE)                   REM   ...
    CALL WRITEF(ZIP)                    REM   ...
    CALL WRITEF(PHONE)                   REM   ...
    ON MATCH(PAYMENT(0),"ABC",1) GOTO ACCT,BANK,PCOD
ACCT:CALL WRITEF("ACCOUNT")
    CALL WRITEF(PAYMENT(1))
    GOTO HCON
BANK:CALL WRITEF("BANK CARD")
    CALL WRITEF(PAYMENT(1))
    GOTO HCON
PCOD:CALL WRITEF("C.O.D.")                REM Cash On Delivery
    RET.ERR% = NXTF(2)
    CALL DM.ERR(RET.ERR%,NXTF$)
HCON:CALL ERR.MSG(75,ON$)                 REM Turn on the exit message
    FOR CNT% = 0 TO ORDER.NO% - 1          REM For each COMPLETED item
      CALL WRITEF(QTY(CNT%))              REM   in order, redisplay QTY,
      CALL WRITEF(DESCRIPTION(CNT%))        REM   description,
      CALL WRITEF(PART.NO(CNT%))           REM   part number,
      CALL WRITEF(PRICE.EA(CNT%))          REM   and unit price.
      RET.ERR% = NXTF(3)                REM Move cursor to Total field
      CALL DM.ERR(RET.ERR%,NXTF$)          REM
```

```
      RET.ERR% = PUTF(TOTAL(CNT%))          REM   and redisplay the total.
    NEXT                        REM Continue for all COMPLETED items
    CALL WRITEF(QTY(ORDER.NO%))             REM For the INCOMPLETE item,
    CALL WRITEF(DESCRIPTION(ORDER.NO%))       REM   redisplay quantity, des-
    CALL WRITEF(PART.NO(ORDER.NO%))          REM cription, and part number.
    CALL SETF(PRM.ON$)                    REM Turn on the prompt
    CALL CURS("0")                      REM Make the cursor visible
    PART.NO(ORDER.NO%) = RESF(1)            REM Replace cursor in field
    CALL SETF(PRM.OFF$)                   REM Turn off the prompt
    GOTO HRET               REM   and resume where entry left off...

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM     If there is no CURRENT.TRM file on the disk...
REM -------------------------------------------------------------------------
ERR1:PRINT "ERROR: No current terminal file"    REM Print error messages
    PRINT "(put control code in 'CURRENT.TRM')"
    STOP                            REM Stop program

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM     If there is no PARTS.LST file on the disk...
REM -------------------------------------------------------------------------
ERR2:PRINT "ERROR: No part no. reference file"  REM Print error message
    STOP                            REM Stop program

REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
REM
REM     NORMAL PROGRAM TERMINATION
REM
REM ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
DONE:CALL SETF ("000000")                   REM Reset terminal attributes
    CALL CLRSCR                       REM Clear the screen.
    RET.ERR% = CLSDIS                     REM Close display file
    CALL DM.ERR(RET.ERR%,CLSDIS$)           REM Check for err during close
    STOP                            REM Return to Operating System
```

Section 3: PL/I-86 user's guide

This section explains how to use Display Manager with application programs written in PL/I-86.

Linking PL/I-86 programs

To link a PL/I-86 program to the Display Manager Run-time Library, use the following command format in response to your Operating System prompt:

```
LINK86 <program name>,DMPLI.L86[S]
```

where <program name> is replaced by the name of the object module produced by the PL/I-86 compiler. For example, to link a program named MYPROG, use the following command line:

```
LINK86 MYPROG,DMPLI.L86[S]
```

PL/I-86 external declarations

PL/I-86 requires that you explicitly declare external functions. External functions are those not coded in the program source code, but referenced by it. The file DMEXTR.PLI contains external declarations for all Display Manager functions. Use the %INCLUDE feature of PL/I-86 to make these external declarations a part of your application program. See the listing at the end of this section for an example.

Function arguments and return values

You must declare numeric values that are used as function arguments, or that are returned to your application program, as 15-bit, signed, fixed-length, binary variables.

You must declare character values that are used as function arguments, or that are returned to your application program, as character varying variables.

The following example illustrates how you should declare function arguments and return values.

```
DECLARE Arg_Int FIXED BINARY(15),       /* Integer */
      Ret_Str CHARACTER(80) VARYING;  /* String */
...
Ret_Str = RESF (Arg_Int);
...
```

Minimizing data space in PL/I-86 programs

The PL/I-86 compiler compares the arguments for each function call with the original argument definition. If you call a function with a string size other than that of the original string, the compiler allocates enough space to copy the string in the function call. Because the compiler allocates this space for each function call, large amounts of memory space can be used very quickly. To avoid this when calling the PUTF function, assign the argument to a global string before making the call. This assignment ensures that the compiler does not allocate extra space. For example,

```
DCL Glob_Str CHAR(132) VAR;     /* PUTF's argument is also */
                        /* declared as 132 bytes.  */
Glob_Str = Mystring;
CALL PUTF (Glob_Str);          /* No extra space allocated */
...
Glob_Str = "This is text for PUTF";
CALL PUTF (Glob_Str);          /* Again, no extra space */
```

...

This technique is not necessary for the other Display Manager functions, because they require only a few bytes.

When using the INITDM function, declare the string that passes the terminal control codes in the following way:

```
CHAR(254) VAR.
```

This ensures that the PL/I-86 compiler does not create extra space, and saves the copying time. For example,

```
DCL InitStr CHAR(254) VAR;
...
/* Put control codes in INITSTR here... */
...
IF InitDM(InitStr) < 0 THEN
  GOTO Err_Cond;
...
```

Sample program (SAMPLI.PLI) listing

The following listing is the source code for a sample program written in PL/I-86. Your distribution disks provide the code in the file named SAMPLI.PLI. This listing is for reference only. Always consider the program on your distribution disks as the definitive version.

Listing 3-1. SAMPLI.PLI source code

```
program:  PROC   options(main);

  %replace onn by '0';     /* Make a field visible. */
  %replace off by '1';     /* Make a field invisible. */
  %replace lst_sz by 50;     /* size of parts list */

  /* Include the Display Manager runtime library definitions. */
  %include 'dmextr.pli';
                     /*     12345678901234  */
dcl  tabs static char(14) init('             '); /*tabs for output */

dcl
    (phone_order,
     order_no,
     page,
     part_fb15,
     cnt,
     CLRSCR_ret,
     ret,
     ret_err,
     helpf)     fixed;
```

```
dcl
   (qty_dec,     /* quantity of each item */
    price_ea_dec)      /* normal price output, but there may be a sale */
        fixed dec (6,2);  /* max is 9,999.99  */
dcl
    total_dec(0:4)         /* QTY times PRICE_EA */
       fixed dec (7,2);     /* max is 99,999.99 */
dcl
    sale_dec  fixed dec (8,2);   /* max is 999,999.99  */

dcl term250_str  char (250) var;

dcl putf132_str char(132) var;

dcl
   (initdm_str,
    temp,
    retf60_str,
    dispd_str,
    opndis_str,
    posf_str,
    nxtf_str,
    putf_str,
    cur_str,
    CLSDIS_str,
    customer,   /* customer name */
    address,
    city,
    state,
    zip,       /* validated for numerical value */
    phone,        /* numerical */
    payment(0:1),    /* method of payment and account no. */
    qty(0:4),     /* quantity of each item */
    price_ea(0:4),      /* normal price output, but there may be a sale */
    total(0:4),         /* QTY times PRICE_EA */
    sale,   /* max is 999,999.99  */
    description(0:4),    /* brief written description */
    part_no_chr60(0:4),      /* 5-digit number, checked for validity */
    part_lst(0:lst_sz,0:1),
    price(lst_sz))            char(60) var;     /* All data entered w/ GETF &
                                        UPDF is in string form. */

 dcl
   (prm_off,
    prm_on,
    retf16_str,
    avail_attr)
             char(16) var;

dcl
   curstat char(1) var ;

dcl
```

```
   (file_1,
    file_2)                file;


/*****************************************************************************/
/* The following corresponds to lines  34- 64 in the CB-80 sample program.   */
/*****************************************************************************/

  /* Get the screen-handling control code from the installation file. */
  on undefinedfile (file_1)
     go to err1;   /* no term file, abort */
  on endfile (file_1) go to err1;
  open file (file_1) stream input title('current.trm');
  get file (file_1) edit(term250_str) (a);
  close file (file_1);

  /* Set up the list of part numbers. */
  on undefinedfile (file_2) go to err2;   /* no input file, abort */
  open file(file_2) stream input title('parts.lst');
  on endfile (file_2) go to s_ls;
  do cnt=0 to lst_sz-1;
    get file(file_2) list(part_lst(cnt,0),part_lst(cnt,1),price(cnt));
 end;

s_ls:
  part_lst(cnt+1,0)='';    /* indicates end of list */
  close file(file_2);

  /* Assign display numbers. These can be changed as needed. */
  phone_order=1;     /* main display */
  helpf=2;      /* main part number reference */

  /* Set error output messages for fatal Display Manager errors. */
  initdm_str='ERROR: Initialization failure';
  opndis_str='ERROR: Display file not found';
  dispd_str='ERROR: Display not found';
  posf_str='ERROR: Field missing';
  nxtf_str='ERROR: Next field missing';
  putf_str='ERROR: Write to field failure';
  cur_str='ERROR: Cursor On/Off failure';
  CLSDIS_str='ERROR: Can''t close display file';


/*****************************************************************************/
/* The following corresponds to lines  65-122 in the CB-80 sample program.   */
/*****************************************************************************/

  /* Most DM functions return -1 if there is an error. */
  /* They are fatal, so abort. */
  dm_err:  PROC  (f_ret,err_type);
    dcl
      err_type char(60) var,
      f_ret fixed;
    if f_ret>=0 then
```

```
    return;          /* not an error */
  put skip(2) list(err_type);
  stop;     /* It's fatal, so abort. */
end dm_err;

/* If the part number exists, return it. */
search:  PROC  (part_no_chr60) returns(fixed);
  dcl
    part_no_chr60  char(60) var;
  do cnt=0 to lst_sz-1;   /* returns the array index */
    if part_lst(cnt,0)=part_no_chr60 then
      return(cnt);
    if part_lst(cnt,0)='' then
      go to elst;
  end;
  elst:
  return(-1);
end search;

/* Move relative to the next input field, turn on the prompt, & get input. */
get_entry:    PROC   returns(char(60) var);
  dcl inp60_local   char(60) var;
  ret_err=nxtf(2);    /* next input field */
  call dm_err(ret_err,nxtf_str);
  avail_attr=setf(prm_on);     /* Turn on the prompt. */
  inp60_local=getf();     /* Input from the field. */
  cont:
  if endf()=27 then     /* escape key to exit */
    do;
      CLRSCR_ret = CLRSCR();    /*added11-8*/
      ret_err=CLSDIS();
      call dm_err(ret_err,CLSDIS_str);
      stop;
    end;
  if endf()^=0 & endf()^=26 then    /* control character, not ctrl-Z */
    go to retr;       /* Ignore the character and continue. */
  avail_attr=setf(prm_off);      /* Turn off the prompt. */
  return(inp60_local);
retr:
  retf60_str=resf(-1);    /* Save the position. */
  inp60_local=resf(1);    /* Resume input. */
  go to cont;     /* Continue. */
end get_entry;

err_msg:  PROC  (pos,onoff);
  dcl
    pos fixed,
    onoff char(1);
  ret=posf(0);    /* Store the current position. */
  ret_err=posf(pos);
  call dm_err(ret_err,posf_str);
  avail_attr=setf(onoff);     /* Turn the message on/off. */
  ret_err=posf(ret);    /* Return to the original position. */
  call dm_err(ret_err,posf_str);
```

```
  end err_msg;

/*****************************************************************************/
/*    S T A R T         P R O G R A M          H E R E              */
/*****************************************************************************/


/*****************************************************************************/
/* The following corresponds to lines 123-232 in the CB-80 sample program.   */
/*****************************************************************************/

  ret_err=initdm(term250_str);    /* Initialize the library. */
  call dm_err(ret_err,initdm_str);
  avail_attr=retdm();      /* Which CRT attributes are available? */
  if substr(avail_attr,3,1)^='0' then    /* If inverse video is supported */
    do;
      prm_on='031';
      prm_off='330';    /* then use it for prompts */
    end;
  else
    do;
      prm_on='0';
      prm_off='3';    /* just initials */
    end;


  /* Open the display file, show it, and move to the first field. */
  ret_err=opndis('ORDERS.DIS');      /* Open the file. */
  call dm_err(ret_err,opndis_str);
loop:
  ret_err=dispd(phone_order);          /* Show the display. */
  call dm_err(ret_err,dispd_str);
  ret_err=nxtf(-10);              /* 1st field */
  call dm_err(ret_err,nxtf_str);

  /* All prompts are inverse video if possible, or underlined otherwise. */
  customer=get_entry();         /* Use relative movement */
  address=get_entry();         /* and GETF */
  city=get_entry();
  state=get_entry();       /* alphabetic only */
  zip=get_entry();      /* numerical validation by DM */
  phone=get_entry();
paym:
  payment(0)=get_entry();    /* A, B, or C only */
                        /* A null string is also not a valid entry. */
  if index('ABC',payment(0))=0 ! payment(0)='' then
    do;
      call err_msg(100,onn);
      ret_err=nxtf(-2);
      call dm_err(ret_err,nxtf_str);    /* Output an error message */
      go to paym;                    /* and re-try. */
    end;
  else
    do;
```

```
      ret_err=nxtf(3);
      call dm_err(ret_err,nxtf_str);    /* Go to next column. */
     end;
   if payment(0)='A' then
    do;  /* It's a personal credit account. */
      putf132_str = 'CCOUNT';
      ret_err=putf(putf132_str);   /* Show the rest of the word. */
      payment(1)=get_entry();    /* Get the account number. */
     end;
   else
    if payment(0)='B' then    /* bank credit card */
     do;
       putf132_str = 'ANK CARD';
       ret_err=putf(putf132_str);
       call dm_err(ret_err,putf_str);
       payment(1)=get_entry();
      end;
    else
     if payment(0)='C' then    /* cash on delivery */
       do;
         putf132_str = '.O.D.';
         ret_err=putf(putf132_str);
         call dm_err(ret_err,putf_str);
       end;
   call err_msg(100,off);     /* Turn it off. */

   /* Take the order now. */
   ret_err=posf(75);     /* Turn on the message */
   call dm_err(ret_err,posf_str);     /* about the ending entry. */
   avail_attr=setf('0');
   order_no=0;    /* up to 5 */
 ordr:
   qty(order_no)=get_entry();    /* quantity of items */
   if qty(order_no)='0' then     /* Stop entry. */
    go to ttls;
   description(order_no)=get_entry();
   call err_msg(76,onn);     /* control-Z reference message */
 part_lbl:
   part_no_chr60(order_no)=get_entry();      /* only for this input */
 hret:
   if endf()=26 then    /* control-z for part number refernce display */
    call help();
   part_fb15 = search(part_no_chr60(order_no));   /* complete input */
   if part_fb15 =-1 then    /* not a valid part number */
    do;
      call err_msg(101,onn);
      ret_err=nxtf(-2);
      call dm_err(ret_err,nxtf_str);
      go to part_lbl;  /* Re-try. */
    end;
   call err_msg(101,off);  /* Turn off the error message. */
   call err_msg(76,off);  /* Turn off the control-z message. */
   ret_err=nxtf(2);     /* Write the normal price. */
   call dm_err(ret_err,nxtf_str);
```

```
 putf132_str = price(part_fb15);
 ret_err=putf(putf132_str);
 call dm_err(ret_err,putf_str);
 avail_attr=setf(prm_on);
ptry:
 price_ea(order_no)=updf();    /* If it's a CR, get the initial value. */
 avail_attr=setf(prm_off);
 ret_err=nxtf(3);      /* the field for the total */
 call dm_err(ret_err,nxtf_str);
 qty_dec = qty(order_no);
 price_ea_dec = price_ea(order_no);
 total_dec(order_no) = qty_dec * price_ea_dec;
 total(order_no) = total_dec(order_no);
 total(order_no) = substr(total(order_no),3);
 putf132_str = total(order_no);
 ret_err=putf(putf132_str);
 call dm_err(ret_err,putf_str);
 order_no=order_no+1;    /* Only 5 are allowed. */
 if order_no < 5 then          /* can break w/QTY = 0 */
   go to ordr;
 order_no=4;   /* Only 0 to 4 are allowed. */
ttls:
 sale_dec =0;
 do cnt=0 to order_no;     /* Calculate the total bill. */
   sale_dec = sale_dec + total_dec(cnt);
 end;
 sale = sale_dec;
 sale = substr(sale,3);
 ret_err=posf(26);
 call dm_err(ret_err,posf_str);
 putf132_str = sale;
 ret_err=putf(putf132_str);             /* Write the total sale. */
 do cnt= 0 to order_no;
    total_dec(cnt) = 0;              /* zero out intermediate totals */
    end;
 call dm_err(ret_err,putf_str);
 ret_err=nxtf(20);            /* wait until ready */
 call dm_err(ret_err,nxtf_str);
 avail_attr=setf('0');        /* Turn on the prompt. */
 retf60_str=getf();           /* Wait for a carriage return. */
 if endf()=27 then
   go to done;

 /* output data to file */
 go to loop;       /* next order */


/*****************************************************************************/
/* The following corresponds to lines 233-249 in the CB-80 sample program.   */
/*****************************************************************************/

 writef:  PROC  (out);
   dcl
     out char(60) var;
```

```
    ret_err=nxtf(2);    /* Go to input field. */
    call dm_err(ret_err,nxtf_str);
    avail_attr=setf('0');       /* Turn on the field. */
    putf132_str = out;
    ret_err=putf(putf132_str);  /* Put in the old data. */
    call dm_err(ret_err,putf_str);
    retf16_str=retf();          /* Check if it's a payment. */
    if posf(0) = 8 then         /* Output the rest in the adjoining
                                field. */
      do;
        ret_err=nxtf(3);
        call dm_err(ret_err,nxtf_str);
        putf132_str = substr(out,2,length(out));
        ret_err=putf(putf132_str);
        call dm_err(ret_err,putf_str);
      end;
  end writef;


/*****************************************************************************/
/* The following corresponds to lines 250-310 in the CB-80 sample program.   */
/*****************************************************************************/

help:  PROC  ;          /* Save your place. */
    retf60_str=resf(-1);        /* Show the part number list. */
    curstat = curs(off);
    ret_err=dispd(helpf);
    call dm_err(ret_err,dispd_str);
  firs:
    cnt=0;
    page=22;    /* Write out the list. */
  nxtl:
    ret_err=posf(cnt+1);
    call dm_err(ret_err,posf_str);
    if part_lst(cnt,0) ^= '' then         /* Output to the end of the list. */
      do;
        temp=part_lst(cnt,0)||tabs||part_lst(cnt,1);
        putf132_str = temp;
        ret_err=putf(putf132_str);
        call dm_err(ret_err,putf_str);
        avail_attr =  setf('0');
      end;
    else
      cnt=-1;
    cnt=cnt+1;
    if cnt ^= 0 & cnt < 22 then
      go to nxtl;
ret_err=posf(100);     /* next page, or exit */
    call dm_err(ret_err,posf_str);
    retf60_str=getf();
    if retf60_str=ascii(27) then
      go to reds;    /* escape, return */
    if retf60_str=ascii(26) then
      if cnt=0 then    /* control-Z, next with wrap */
```

```
        go to firs;
     page=page+21;    /* next page */
     cnt = 0;
     go to nxtl;
   reds:      /* bring back old display */
     ret_err=dispd(phone_order);
     call dm_err(ret_err,dispd_str);
     ret_err=nxtf(-10);               /* 1st field, then 1st */
     call dm_err(ret_err,nxtf_str);     /* in field to write   */
     call writef(customer);           /* old data to         */
     call writef(address);
     call writef(city);
     call writef(state);
     call writef(zip);
     call writef(phone);

     if payment(0)  = 'A' then
       go to acct;
     else if payment(0)  = 'B' then go to bank  ;
          else if payment(0)   = 'C' then
            go to pcod  ;

  acct:
     call writef('ACCOUNT');       /* special handling */
     call writef(payment(1));      /* done in writef   */
     go to hcon;
   bank:
     call writef('BANK CARD');
     call writef(payment(1));
     go to hcon;
   pcod:
     call writef('C.O.D.');
     ret_err=nxtf(2);               /* pass acount number */
     call dm_err(ret_err,nxtf_str);
  hcon:
     call err_msg(75,onn);  /* QTY exit message */
     do cnt=0 to order_no-1;     /* Write any       */
       call writef(qty(cnt));    /* previous items. */
       call writef(description(cnt));
       call writef(part_no_chr60(cnt));
       call writef(price_ea(cnt));
       ret_err=nxtf(3);          /* total is output -- */
       call dm_err(ret_err,nxtf_str);  /* field, not input   */
       putf132_str = total(cnt);
       ret_err=putf(putf132_str);
     end;
     call writef(qty(order_no));         /* line in progress */
     call writef(description(order_no));
     call writef(part_no_chr60(order_no));
     avail_attr = setf(prm_on);
     curstat =  curs(onn);
     part_no_chr60(order_no) = resf(1);
     avail_attr = setf(prm_off);
     goto hret;
```

```
  end help;


/*************************************************************************/
/* The following corresponds to lines 311-319 in the CB-80 sample program.   */
/*************************************************************************/

  err1:
  put list('ERROR: No current terminal file');
  put list('(put control code in "CURRENT.TRM")');
  stop;     /* no terminal codes */
  err2:
  put list('ERROR: No part no. reference file');
  stop;     /* no price list -- */
  done:
  CLRSCR_ret =  CLRSCR();
  ret_err=CLSDIS();     /* close display file */
  call dm_err(ret_err,CLSDIS_str);
  stop;

end program;
```

Section 4: Pascal/MT+86 user's guide

This section explains how to use Display Manager with application programs written in Pascal/MT+86. This section also describes special Pascal/MT+86 functions needed to use Display Manager.

Linking Pascal/MT+86 programs

To link a Pascal/MT+86 program to the Display Manager Run-time Library, use one of the following command formats in response to your Operating System prompt:

```
      LINKMT <program name>,DMPAS/S,FULLHEAP,FPREALS/S,PASLIB/S
or
      LINKMT <program name>,DMPAS/S,FULLHEAP,BCDREALS/S,PASLIB/S/X:1000
```

where <program name> is replaced by the name of the object module produced by the Pascal/MT+86 compiler. For example, to link a program named MYPROG, use one of the following command lines:

```
      LINKMT MYPROG,DMPAS/S,FULLHEAP,FPREALS/S,PASLIB/S
or
      LINKMT MYPROG,DMPAS/S,FULLHEAP,BCDREALS/S,PASLIB/S/X:1000
```

You can exclude FULLHEAP if you use stack allocation. The real-number libraries are also optional; the Run-time Library does not have to use FULLHEAP.ERL. If you do not want to use Heap management, a routine is available in the Run-time Library that you can use to allocate space when

needed for your largest display, and when the stack version of NEW/DISPOSE is being used. You can make full use of MARK/RELEASE with this routine.

To simplify the linking process, you can use the librarian LIBMT to create a single, searchable library with DMPASC.ERL, FULLHEAP.ERL, and either FPREALS.ERL or BCDREALS.ERL. In such cases, you need only specify one additional run-time library. For example,

```
LINKMT MYPROG,DMLIBS/S,PASLIBS/S
```

Pascal/MT+86 external declarations

Pascal/MT+86 requires that you explicitly declare external functions. External functions are those not coded in the program source code, but referenced by it. The file DMEXTR.PAS contains external declarations for all Display Manager functions. Use the Include File compiler toggle of Pascal/MT+86 to make these external declarations a part of your application program. For example, the following statement in your program includes the external declarations:

```
{$I DMEXTR.PAS}
```

Function arguments and return values

Numeric values used as function arguments, or returned to your application program, must be of type integer. Character values used as function arguments, or returned to your application program, must be of type string. Character strings are returned from Display Manager to an extra parameter in the call. The extra parameter must be of type string, because the function returns a character. Consider the following example:

```
VAR
   Arg_Str: STRING;
   Ret_Int: INTEGER;
   Ret_Str: STRING;
   Ret: CHAR
   ...
   Ret_Int := INITDM (Arg_Str)
   ...
   Ret := RESF (1, Ret_Str);
   ...
```

Special Pascal/MT+86 functions

The Display Manager Run-time Library for Pascal/MT+86 contains a number of special functions for your use. These functions are described on the following pages.

INTSTR function

Syntax: <char> := INTSTR (<integer>,<string>)

Explanation:
This function converts an integer to its string value.

Function arguments and return values:
The first argument (<integer>) is the constant or integer variable to be converted. The second argument (<string>) is the string variable to receive the converted value. The converted value is in the form given by WRITE/WRITELN.

Example:
This example returns the string value 123.

```
Ret_Char := INTSTR (123, Str);
WRITELN (Str);
...
```

FPSTR and BCDSTR functions

Syntax: <char> := FPSTR (<floating real>,<string>)
<char> := BCDSTR (<binary real>,<string>)

Explanation:
These functions convert a real number to its string value. You must use the function that corresponds to the real-number library you are using. That is to say, if you are using the FPREALS library, use the FPSTR function; if you are using the BCDREALS library, use the BCDSTR function.

Function arguments and return values:
The first argument (<floating real> or <binary real>) is the constant or real variable to be converted. The second argument (<string>) is the string variable to receive the converted value. The converted value of the number is in the form given by WRITE/WRITELN.

Example:
The following example shows the use of the FPSTR function. This example returns the string 1.23450E+02.

```
...
Ret_Char := FPSTR (123.45, Str);
WRITELN (Str);
...
```

The next example shows the use of the BCDSTR function. This example returns the string 123.4500.

```
...
Ret_Char := BCDSTR (123.45, Str);
```

```
    WRITELN (Str);
    ...
```

INTVAL function

Syntax: <integer> := INTVAL (<string>)

Explanation:
This function converts a string to its integer value.

Function arguments and return values:
The argument (<string>) is the string value of the number to be converted. The integer value of the argument is returned. If the argument contains a real number, it is truncated to the value to the left of the decimal point. If the argument is in E form, an incorrect value is returned.

Example:
The following example returns a value of 1234.

```
    ...
    Int := INTVAL ('1234');
    WRITELN (Int);
    ...
```

The next example returns a value of 1.

```
    ...
    Int := INTVAL ('1.234E+02');
    WRITELN (Int);
    ...
```

FPVAL and BCDVAL functions

Syntax: <floating real> := FPVAL (<string>)
    <binary real> := BCDVAL (<string>)

Explanation:
These functions convert a string to its real-number value. You must use the function that corresponds to the real-number library you are using. That is to say, if you are using the FPREALS library, use the FPVAL function; if you are using the BCDREALS library, use the BCDVAL function.

Function arguments and return values:
The argument (<string>) is the string value of the number to be converted. The value of the argument is returned as a real number. The standard format of the real-number type being used is accepted.

Example:
The first example shows the use of the FPVAL function, and returns the real number 1.23450E+02.

```
...
Real := FPVAL ('123.45');
WRITELN (Real);
...
```

The next example shows the use of the BCDVAL function, and returns the real number 123.4500.

```
...
Real := BCDVAL ('123.45');
WRITELN (Real);
...
```

DMALLO function

Syntax: <integer> := DMALLO (<integer>)

Explanation:
The DMALLO function allocates space for displays when FULLHEAP.ERL is not being used. If you use FULLHEAP.ERL, you should not use DMALLO, because space is automatically allocated.

Function argument and returned values:
The argument (<integer>) is a number ranging from 1 to 6. The number indicates how many Kilobytes are to be allocated, and must be sufficient to accomodate the largest display used in your program (5KB is almost always sufficient). If MARK and RELEASE are used in your program and you release space allocated with DMALLO, you must call the DMALLO function again before using the DISPD function.

If the argument value is less than 1, or greater than 6, a value of -1 is returned to your application program. If no space is available for the DMALLO function, a run-time error results.

Example:

```
...
Int := DMALLO (5);
Err_Ret := DISPD (1);
...
```

Sample program (SAMPAS.PAS) listing

The following listing is the source code for a sample program written in Pascal/MT+86. Your distribution disks provide the code in the file named SAMPAS.PAS. This listing is for reference only. Always consider the code on your distribution disks as the definitive version of the program.

The sample Pascal/MT+86 program must be compiled and linked for use with BCD

numbers, as follows:

```
MT86 SAMPAS $B
LINKMT SAMPAS,DMPAS/S,BCDREALS,FULLHEAP,PASLIB/S/X:100
```

If you encounter problems using the special Display Manager functions BCDVAl and BCDSTR, request an updated version of the Pascal/MT+86 library BCDREALS.L86 from Digital Research.

Listing 4-1. SAMPAS.PAS source code

```
program sample;
const
  onn = '000';     { Make a field visible. }
  off = '100';     { Make a field invisible. }
  lst_sz = 25;     { size of parts list }
  { Assign display numbers. These can be changed as needed. }
  phone_order = 1;     { main display }
  helpf = 2;    { main part number reference }

  { Set error output messages for fatal Display Manager errors. }
  initdm_str = 'ERROR: Initialization failure';
  opndis_str = 'ERROR: Display file not found';
  dispd_str = 'ERROR: Display not found';
  posf_str = 'ERROR: Field missing';
  nxtf_str = 'ERROR: Next field missing';
  putf_str = 'ERROR: Write to field failure';
  cur_str = 'ERROR: Cursor On/Off failure';
  CLSDIS_str = 'ERROR: Can''t close display file';

    {     12345678901234  }
  tabs = '             '; {tabs for output }

type
     com_str = string[40];
     ptr = ^integer;
var
     order_no,
     page,
     part_fb15,
     cnt,
     CLRSCR_ret,
     ret,
     ret_err    : integer;

     retchr     : char;

     qty_dec,                    { quantity of each item }
     price_dec: real;          { normal price given, but may be sale }

     total_dec  : array[0..4] of real;  { QTY times PRICE_EA }
```

```
    sale_dec   : real;

    term250_str: string[250];

    retf60_str,
    customer,                    { customer name }
    address,
    city,
    state,
    zip,                        { validated for numerical value }
    phone,                       { numerical }
    sale      : com_str;
    qty,                        { quantity of each item }
    price_ea,                    { normal price shown, but may be sale }
    total,                      { QTY times PRICE_EA }
    description,                 { brief written description }
    part_no_chr60: array[0..4] of com_str;
    part_lst   : array[0..lst_sz,0..1] of com_str;
    price      : array[0..lst_sz] of com_str;
    payment    : array[0..1] of com_str;{ method of payment and account no. }
    buff_rd    : string[60];
    temp       : string;

    prm_off,
    prm_on,
    retf16_str,
    avail_attr : string[16];

    curstat    : string[1];

    file_1,
    file_2     : text;


  { Include the Display Manager runtime library definitions. }
{$I dmextr.pas}
external procedure @hlt;


{*************************************************************************}
{ The following corresponds to lines  65-122 in the CB-80 sample program.   }
{*************************************************************************}

procedure halt;
begin
   @hlt;                        { stop the program }
end;

  { Most DM functions return -1 if there is an error. }
  { They are fatal, so abort. }
procedure dm_err(f_ret : integer;err_type : com_str);
begin
    if f_ret < 0
     then begin
```

```
      writeln; writeln;
      writeln(err_type);
      halt;                    { It's fatal, so abort. }
    end;
end; { dm_err }

  { If the part number exists, return it. }
function search(part_no_chr60 : com_str) : integer;
var cnt : integer;
begin
  for cnt := 0 to lst_sz-1 do          { returns the array index }
   begin
    if part_lst[cnt,0] = part_no_chr60
     then begin
       search :=  cnt;
       exit;
     end;
    if part_lst[cnt,0] = ''
     then begin
       search :=  -1;                        { -1 unless found }
       exit;
     end;
   end;
  search :=  -1;                         { -1 unless found }
end; { search }

  { Move relative to the next input field, turn on the prompt, & get input. }
procedure get_entry(var retval : com_str);
var inp60_local : com_str;
begin
  ret_err :=  nxtf(2);               { next input field }
  dm_err(ret_err,nxtf_str);
  retchr := setf(prm_on,avail_attr);    { Turn on the prompt. }
  retchr := getf(inp60_local);         { Input from the field. }
  while true do begin
   if endf = 27
    then begin                     { escape key to exit }
      CLRSCR_ret := clrscr;          {added11-8}
      ret_err := clsdis;
      dm_err(ret_err,CLSDIS_str);
      halt;
    end;
   if (endf <> 0) and (endf <> 26)
    then begin                     { control character, not ctrl-Z }
      retchr := resf(-1,retf60_str);  { Save the position. }
      retchr := resf(1,inp60_local);  { Resume input. }
    end else begin
      retchr := setf(prm_off,avail_attr);      { Turn off the prompt. }
      retval := inp60_local;
      exit;
    end;
  end;
end; { get_entry }
```

```
procedure err_msg(pos : integer;onoff : string);
begin
  ret := posf(0);                  { Store the current position. }
  ret_err := posf(pos);
  dm_err(ret_err,posf_str);
  retchr := setf(onoff,avail_attr);     { Turn the message on/off. }
  ret_err := posf(ret);               { Return to the original position. }
  dm_err(ret_err,posf_str);
end; { err_msg }


{*****************************************************************************}
{ The following corresponds to lines 233-249 in the CB-80 sample program.   }
{*****************************************************************************}

procedure writef(out : com_str);
begin
  ret_err := nxtf(2);                { Go to input field. }
  dm_err(ret_err,nxtf_str);
  retchr := setf(onn,avail_attr);       { Turn on the field. }
  ret_err := putf(out);               { Put in the old data. }
  dm_err(ret_err,putf_str);
  retchr := retf(retf16_str);          { Check if it's a payment. }
  if posf(0) = 8
    then begin                     { Output rest in adjoining field. }
      ret_err := nxtf(3);
      dm_err(ret_err,nxtf_str);
      ret_err := putf(copy(out,2,length(out)-1));
      dm_err(ret_err,putf_str);
    end;
end; { writef }


{*****************************************************************************}
{ The following corresponds to lines 250-310 in the CB-80 sample program.   }
{*****************************************************************************}

procedure help;
var cnt : integer;
begin
  while endf = 26 do begin
    retchr := resf(-1,retf60_str);      { Show the part number list. }
    retchr := curs(off,curstat);
    ret_err := dispd(helpf);
    dm_err(ret_err,dispd_str);
    retf60_str := chr(0);
    cnt := 0;
    page := 22;                    { Write out the list. }
    repeat
      ret_err := posf(cnt+1);
      dm_err(ret_err,posf_str);
      if part_lst[cnt,0] <> ''
        then begin                 { Output to the end of the list. }
          temp := concat(part_lst[cnt,0],tabs,part_lst[cnt,1]);
```

```
      ret_err := putf(temp);
      dm_err(ret_err,putf_str);
      retchr := setf(onn,avail_attr);
    end else cnt := -2;
  cnt := cnt+1;
  if (cnt = -1) or (cnt >= 22)
    then begin
      ret_err := posf(100);        { next page, or exit }
      dm_err(ret_err,posf_str);
      retchr := getf(retf60_str);
      if retf60_str <> chr(27)
        then begin
          if retf60_str = chr(26)
            then if cnt <> -1
              then begin          { control-Z, next with wrap }
                page := page+21;    { next page }
                cnt := 0;
              end else begin
                cnt := 0;
                page := 22;
        end;
      end;
  end;
until retf60_str = chr(27);
ret_err := dispd(phone_order);
dm_err(ret_err,dispd_str);
ret_err := nxtf(-10);                  { 1st field, then 1st }
dm_err(ret_err,nxtf_str);          { in field to write   }
writef(customer);                 { old data to        }
writef(address);
writef(city);
writef(state);
writef(zip);
writef(phone);
case payment[0,1] of
  'A' : begin
    writef('ACCOUNT');             { special handling }
    writef(payment[1]);            { done in writef   }
    end;
  'B' : begin
    writef('BANK CARD');
    writef(payment[1]);
    end;
  'C' : begin
    writef('C.O.D.');
    ret_err := nxtf(2);           { pass acount number }
    dm_err(ret_err,nxtf_str);
    end;
end;
err_msg(75,onn);                  { QTY exit message }
for cnt := 0 to order_no-1 do              { Write any        }
  begin
    writef(qty[cnt]);             { previous items. }
    writef(description[cnt]);
```

```
      writef(part_no_chr60[cnt]);
      writef(price_ea[cnt]);
      ret_err := nxtf(3);           { total is output -- }
      dm_err(ret_err,nxtf_str);     { field, not input   }
      ret_err := putf(total[cnt]);
     end;
   writef(qty[order_no]);        { line in progress }
   writef(description[order_no]);
   writef(part_no_chr60[order_no]);
   retchr := setf(prm_on,avail_attr);
   retchr := curs(onn,curstat);
   retchr := resf(1,part_no_chr60[order_no]);
   retchr := setf(prm_off,avail_attr);
 end; { while }
end; { help }


{*****************************************************************************}
{ The following corresponds to lines  34- 64 in the CB-80 sample program.   }
{*****************************************************************************}

{ The errors below correspond to lines 311-319 in the CB-80 sample program. }

procedure init_data;
begin
 { Get the screen-handling control code from the installation file. }
 open(file_1,'current.trm',ret_err);
 if ret_err <> 255
   then begin
     readln(file_1,term250_str);
     if ioresult <> 0
       then ret_err := 255;
   end;
 if ret_err = 255
   then begin
     writeln('ERROR: No current terminal file');
     writeln('(put control code in "CURRENT.TRM")');
     halt;                         { stop }
   end;
 { Set up the list of part numbers. }
 open(file_2,'parts.lst',ret_err);
 if ret_err <> 255
   then begin
     cnt :=  0;
     while (not eof(file_2)) and (cnt < lst_sz) do
       begin
         readln(file_2,buff_rd);
         part_lst[cnt,0] := copy(buff_rd,1,5);
         buff_rd[6] := ' ';
         page := pos(',',buff_rd);
         part_lst[cnt,1] := copy(buff_rd,8,page-9);
         price[cnt] := copy(buff_rd,page+1,length(buff_rd)-page);
         cnt :=  cnt+1;
       end;
```

```
    end else begin
      writeln('ERROR: No part no. reference file');
      halt;                         { stop }
    end;
  part_lst[cnt+1,0] :=  '';    { indicates end of list }
  close(file_2,ret_err);
  close(file_1,ret_err);
end; { init_data }

procedure head;
begin
    { All prompts are inverse video if possible, or underlined otherwise. }
    get_entry(customer);              { Use relative movement }
    get_entry(address);               { and GETF }
    get_entry(city);
    get_entry(state);                 { alphabetic only }
    get_entry(zip);                   { numerical validation by DM }
    get_entry(phone);
    get_entry(payment[0]);             { A, B, or C only }
                                { null string not a valid entry. }
    while (pos(payment[0],'ABC') = 0) or (payment[0] = '') do
      begin
        err_msg(100,onn);
        ret_err := nxtf(-2);
        dm_err(ret_err,nxtf_str);       { Output an error message }
        get_entry(payment[0]);          { retry }
      end;
    ret_err := nxtf(3);
    dm_err(ret_err,nxtf_str);           { Go to next column. }
    case payment[0,1] of
      'A' : begin                   { It's a personal credit account. }
        ret_err := putf('CCOUNT');      { Show the rest of the word. }
        get_entry(payment[1]);          { Get the account number. }
        end;
      'B' : begin                   { bank credit card }
        ret_err := putf('ANK CARD');
        dm_err(ret_err,putf_str);
        get_entry(payment[1]);
        end;
      'C' : begin                   { cash on delivery }
        ret_err := putf('.O.D.');
        dm_err(ret_err,putf_str);
        end;
    end;
    err_msg(100,off);                 { Turn it off. }
end; { head }


begin { program }


{*************************************************************************}
{    S T A R T         P R O G R A M          H E R E                }
{*************************************************************************}
```

```
{*************************************************************************}
{ The following corresponds to lines 123-232 in the CB-80 sample program.   }
{*************************************************************************}

 init_data;
 ret_err := initdm(term250_str);       { Initialize the library. }
 dm_err(ret_err,initdm_str);
 retchr := retdm(avail_attr);          { Which CRT attributes are available? }
 if avail_attr[3] <> '0'
  then begin                        { If inverse video is supported }
    prm_on := '031';
    prm_off := '330';                 { then use it for prompts }
  end else begin
    prm_on := '0';
    prm_off := '3';                   { just initials }
  end;


 { Open the display file, show it, and move to the first field. }
 ret_err := opndis('ORDERS.DIS');      { Open the file. }
 dm_err(ret_err,opndis_str);
 repeat
  ret_err := dispd(phone_order);      { Show the display. }
  dm_err(ret_err,dispd_str);
  ret_err := nxtf(-10);               { 1st field }
  dm_err(ret_err,nxtf_str);
  head;

  { Take the order now. }
  ret_err := posf(75);                { Turn on the message }
  dm_err(ret_err,posf_str);           { about the ending entry. }
  retchr := setf(onn,avail_attr);
  order_no := 0;                    { up to 5 }
  repeat
    get_entry(qty[order_no]);         { quantity of items }
    qty_dec := BCDVAL(qty[order_no]);
    if qty_dec <> 0
     then begin                     { Stop entry. }
       get_entry(description[order_no]);
       err_msg(76,onn);               { control-Z reference message }
       repeat
         get_entry(part_no_chr60[order_no]);
         help;                     { ^Z gives part # display }
         part_fb15 := search(part_no_chr60[order_no]);
         if part_fb15  = -1
           then begin               { not a valid part number }
             err_msg(101,onn);
             ret_err := nxtf(-2);
             dm_err(ret_err,nxtf_str);
           end;
       until part_fb15 <> -1;        { retry }
       err_msg(101,off);             { Turn off the error message. }
       err_msg(76,off);              { Turn off the control-z message. }
```

```
        ret_err := nxtf(2);          { Write the normal price. }
        dm_err(ret_err,nxtf_str);
        ret_err := putf(price[part_fb15]);
        dm_err(ret_err,putf_str);
        retchr := setf(prm_on,avail_attr);
        retchr := updf(price_ea[order_no]);{ If CR, get the initial value. }
        retchr := setf(prm_off,avail_attr);
        ret_err := nxtf(3);          { the field for the total }
        dm_err(ret_err,nxtf_str);
        price_dec := BCDVAL(price_ea[order_no]);
        total_dec[order_no] := qty_dec * price_dec;
        retchr := BCDSTR(total_dec[order_no],total[order_no]);
        ret_err := putf(total[order_no]);
        dm_err(ret_err,putf_str);
        order_no := order_no+1;      { Only 5 are allowed. }
      end;
    until (qty_dec = 0) or (order_no >= 5);
    order_no := 4;                   { Only 0 to 4 are allowed. }
    sale_dec := 0;
    for cnt := 0 to order_no do { Calculate the total bill. }
      sale_dec := sale_dec + total_dec[cnt];
    retchr := BCDSTR(sale_dec,sale);
    ret_err := posf(26);
    dm_err(ret_err,posf_str);
    ret_err := putf(sale);             { Write the total sale. }
    for cnt := 0 to order_no do
      total_dec[cnt] := 0;             { zero out intermediate totals }
    dm_err(ret_err,putf_str);
    ret_err := nxtf(20);               { wait until ready }
    dm_err(ret_err,nxtf_str);
    retchr := setf(onn,avail_attr);    { Turn on the prompt. }
    retchr := getf(retf60_str); { Wait for a carriage return. }
  until endf = 27;
  { output data to file }
  clrscr_ret := clrscr;
  ret_err := clsdis;                 { close display file }
  dm_err(ret_err,clsdis_str);
  exit;
end.


EOF
```

DMPGPC.WS4 (= Display Manager Programmer's Guide)

- "Display Manager Programmer's Guide"
  for PC DOS

First Edition: July 1983

(Retyped by Emmanuel ROCHE.)

--> To be found... <--

DMRM.WS4 (= Display Manager Reference Manual)

- "Display Manager Reference Manual"

First Edition: July 1983

(Retyped by Emmanuel ROCHE.)

Foreword

Display Manager, a productivity tool from Digital Research, is a quick, easy-to-use, efficient tool to help you design and use display screens in your application programs.

For system designers, Display Manager offers the ability to design display screens exactly as they appear to the end-user of a program. Display design takes place on a computer terminal screen, not on paper worksheets.

For programmers, Display Manager simplifies programming tasks, reduces the size and complexity of programs, and makes it possible for a program to work on many different computer terminals with virtually no changes required in the program code. In many ways, Display Manager is an automatic programming tool.

Your Display Manager documentation includes the following:

1) "Display Manager Reference Manual"
2) "Display Manager Programmer's Guide"

Your "Reference Manual" describes how to create displays and use them in your application programs. Your "Programmer's Guide" contains information specific to using Display Manager with a particular operating system and programming languages. You need both the "Reference Manual" and "Programmer's Guide" to make proper use of Display Manager.

Note: Due to the limitations of the hardware of 8-bit microcomputers, the Help and Color facilities in Display Manager are not available to computers based on these chips. If you are using Display Manager on such equipment, disregard all references in your documentation to these facilities.

Table of Contents
---

6 Editor Commands

Cursor Movement Commands

- Beginning of Field: Ctrl-US or Ctrl-UH
- Beginning of Next Line: RETURN
- Down Half Screen: Ctrl-QX or Ctrl-QJ
- Down One Line: Ctrl-X or Ctrl-J
- End of Field: Ctrl-UD or Ctrl-UL
- Left Half Screen: Ctrl-QS or Ctrl-QH
- Left One Space: Ctrl-S or Ctrl-H
- Next Field: Ctrl-UF
- Next Word: Ctrl-F
- Previous Field: Ctrl-UA
- Previous Word: Ctrl-A
- Right Half Screen: Ctrl-QD or Ctrl-QL
- Right One Space: Ctrl-D or Ctrl-L
- Tab: Ctrl-I
- Up Half Screen: Ctrl-QE or Ctrl-QK
- Up One Line: Ctrl-E or Ctrl-K

Field Editing Commands

- Boundary Display (All Fields): Ctrl-QB
- Boundary Display (Fields on a Single Line): Ctrl-B
- Change Field to Literal: Ctrl-UZ
- Copy Field to Cursor Location: Ctrl-UC
- Define Input Field: Ctrl-UI
- Define Output Field: Ctrl-UO
- Delete Field: Ctrl-UG
- Move Right Field: Ctrl-UV
- Move Field to Cursor Location: Ctrl-UM
- Renumber Fields: Ctrl-UR

Tables

Figures

Section 1: Introduction to Display Manager

A time-consuming, tedious task in developing computer programs is designing, creating, and maintaining the displays that show on the screen at the time the program is run. For example, if you write a program in CBASIC Compiler, you need a large number of PRINT and PRINT USING statements to show displays on the screen. If the program has many displays, it might require hundreds of such statements. Furthermore, the displays require a large amount of main memory, and make the program difficult to debug and maintain.

Display Manager solves these and other problems by making it possible to design displays directly on your terminal screen. When a display looks exactly as you want it to appear when your program is run, you can store it in a disk file for subsequent use by the program. When the program needs to show a

display on the terminal, it reads the display from the disk file, and places it on the screen.

You can design displays on one terminal, then show them on the same or different terminals. Display Manager makes it possible to design displays that work properly on a wide variety of terminals.

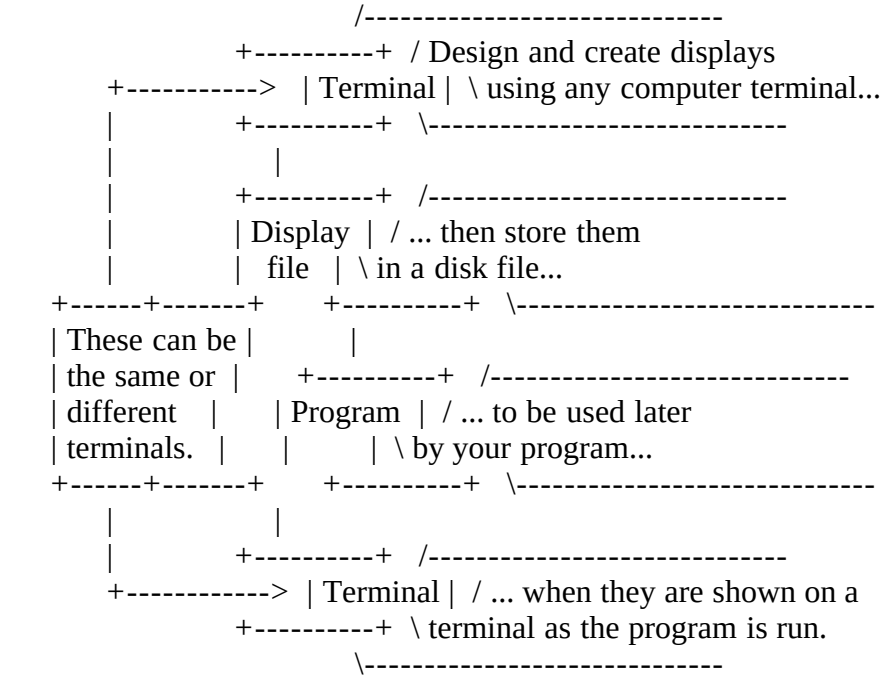
1.1 Major components

Display Manager has 3 major components:

1) The Terminal Setup Program makes it possible for your application programs to work with whatever computer terminals you require. Under most circumstances, simply run the program, and select the terminals that you want to use from a list.

2) The Editor Program helps you design, create, change, and delete displays directly on your terminal screen. Many additional options are available to you in this program.

3) The Run-time Library is a library of routines that your application programs can use to manage and manipulate the displays created with the Editor program.



(Displays can be designed to work on a wide variety of terminals.)

Figure 1-1. Basic Principles

1.2 Benefits

The following sections describe the most significant benefits of using Display Manager.

1.2.1 What You See Is What You Get

You design displays with Display Manager on a terminal screen to look exactly as you want them to appear when your application program runs. Display Manager also simplifies using available features on a terminal, such as highlighting, inverse video, underlining, color, and others.

1.2.2 Reduced Program Size

Programs written using Display Manager require fewer lines of code. Here are the primary reasons:

- You can virtually eliminate PRINT- and PRINT USING-type statements, replacing them with a single statement calling a Display Manager routine to perform the same functions.

- Display Manager routines provide the necessary logic to check the validity of information entered on a terminal by an end-user. Consequently, your programs do not need extensive data validation routines. Display Manager's validation routines provide you with several options for handling invalid input.

- The actual image of each display in a program is stored in a compacted file on disk, and not as part of your program. This reduces the size of both the source and object program; consequently, the program requires less memory to run.

1.2.3 Easier Program Maintenance

Because your programs contain fewer lines of code, they are shorter, simpler, and much easier to debug and maintain. In many programs, PRINT or PRINT USING and data validation routines alone comprise much of the code. Their elimination reduces the complexity and size of your programs.

1.2.4 Simplified Display Designing Methods

Without Display Manager, the simplest method for designing displays is to lay them out on worksheets as a means of determining the row and column numbers for each field. With Display Manager, you type the information that you want shown in your display on the screen of your terminal. You reserve space in the display for entering information, and displaying information derived by your program. Row and column numbers are no longer a major concern.

Also, changes in the design of a display do not require poring over endless formatting statements, such as PRINT USING, to find and change the correct ones. Simply use the Display Manager Editor to place the display back on your terminal screen, make the necessary changes, and store the modified display back in the disk file.

1.2.5 Automatic Documentation

By entering a single command on your terminal keyboard, you can instruct Display Manager to prepare detailed documentation for each display you create. You can either print the documentation, or store it in a disk file. You can then use a word processor to enhance the information as required. This is a simple, effective method for creating user manuals and program documentation.

1.2.6 Uses the Features of any Computer Terminal

Without Display Manager, creating an application program to work with a variety of terminals is a complex task. You must determine what codes to send to the terminal to activate its features, then code them into your program. Because terminals vary significantly in their features and codes, this can add a great deal of time and difficulty to the task.

With Display Manager, creating programs to work with different terminals requires minimum effort on your part. In most cases, you only must enter one line of code in your source program to make it work with different terminals. This is even true when you design applications to work on both monochrome and color terminals.

1.2.7 Separates Designing and Programing Tasks

With Display Manager, designing displays is independent of coding the program. A systems analyst can develop as a separate step all displays for an application. A programmer can then use the displays, specifications, and documentation created with Display Manager to do the coding in the most efficient manner. This separation of tasks means that the user-interface for your applications is designed and created independently of the actual program logic, thus making your applications more user-oriented.

1.2.8 Optimum Response Times

Display Manager handles screen display, validation of input information, and output formatting using optimized assembly code. This provides response times unattainable by other methods, and frees you from these basic tasks.

Many additional benefits become apparent as you begin to use Display Manager.

Section 2: How Display Manager Works

This section explains how Display Manager works by discussing the various environments in which it functions. For this discussion, environment refers to your computer hardware and software. Because several terms and concepts unique to Display Manager first appear in this section, you might want to study the Glossary before reading this material.

You use Display Manager in 4 primary environments:

1) Terminal Setup
2) Editor
3) Applications Programming
4) Run-time

A brief discussion of each environment follows. Sections 3 through 8 contain detailed descriptions of each environment and its options.

2.1 TERMINAL SETUP ENVIRONMENT

Display Manager works with most terminals on the market today. Because these terminals vary in their features, capabilities, and operation, you must specify to Display Manager which terminals you will use. The terminal setup program, named DMSET, helps you do this.

Display Manager sends control codes to a terminal to activate it and use its features. Therefore, to function properly with a terminal, Display Manager must know what control codes to send. On your distribution disks is a file named TERMS.DM, which contains the control codes for those terminals that you can use with Display Manager.

When run, the DMSET program shows you a list of the terminals in the TERMS.DM file, and asks you to pick the terminal or terminals that you want to use from that list. In most cases, that is the only step needed to set up a terminal for use with Display Manager (see Section 3 for more detailed information). If you want to use Display Manager with a terminal not in TERMS.DM, you must provide the control codes for that terminal by answering a series of questions that the DMSET program asks. These questions are in common English but, to answer them, you need the manual for the terminal in question (see Appendix C for more detailed information).

A terminal used with Display Manager can fall into one, or both, of two categories:

1) A design terminal serves to actually create the displays for your application program. You use this terminal with the Editor program in the Editor environment. The Editor program can only be set up to use one, specific design terminal, but you can set up different versions of the Editor for use with other design terminals.

2) Run-time terminals function at the time the application program runs, to present the displays created on the design terminal. You can use run-time terminals in the run-time environment. You can design application programs to work with any terminal in the TERMS.DM file.

Section 3 and Appendix C describe the options available in the terminal setup environment. Appendix A describes terminal control codes and the TERMS.DM file.

2.2 THE EDITOR ENVIRONMENT

In this environment, you use the design terminal to create the displays that your application program uses in the run-time environment.

While using the Editor, you create a display directly on the screen of the design terminal. When the display is exactly as you want it to appear at run-time, you store that display in a display file on disk. Your application program can use the displays in this file at run-time.

Besides creating displays, you can use the Editor to change displays after they are created, remove displays from the file when they are no longer needed, prepare documentation for individual displays, and more.

Sections 4 through 6 describe how to use the Editor.

2.3 APPLICATIONS PROGRAMMING ENVIRONMENT

Once you have created the displays that you want to use at run-time, you can write your application program using one of the Digital Research programming languages that works with Display Manager (refer to your "Display Manager Programmer's Guide").

As you create the source code for your program, you code in function calls to Display Manager routines. These routines are in the Display Manager Run-time Library, and provide the logic needed to do these and other things:

- Place a display on the run-time terminal screen.

- Retrieve information entered by the end-user.

- Show information on the display.

- Place the cursor in a specific location in the display.

- Activate or deactivate features on the run-time terminal, such as inverse video, half intensity, color, and graphics.

After your application program is compiled using the appropriate Digital Research compiler, link the resulting object module to the Run-time Library to include the necessary routines as part of your program. You can then use the

object module, with the included routines, in the run-time environment.

Section 7 describes the applications programming environment. It also includes descriptions of the various Display Manager routines (functions) that you can use in your application programs.

2.4 RUN-TIME ENVIRONMENT

Everything created and accomplished in the 3 preceding environments comes together in the run-time environment. This includes the following:

- the file of terminal control codes created with the DMSET program

- the display file created with the Editor containing the displays to be used by your application program

- your application program object module including the routines linked in from the Run-time Library

In the run-time environment, the end-user runs your program using a run-time terminal. Your program retrieves displays from the display file, and shows them on the run-time terminal screen.

Section 8 describes the run-time environment. It also lists and describes possible run-time errors.

2.5 SUMMARY

To summarize, here are the steps usually required to use Display Manager:

1. Create a version of the Editor program for use with your particular design terminal. This step need only be completed once for a particular design terminal, and is accomplished using the DMSET program.

2. Specify the terminals with which your application program will be used by setting up the control codes for each. Use the DMSET program to accomplish this step, too.

3. Create the displays that you will use in your application program using the version of the Editor program created in step 1. Prepare documentation for your displays using an option available in the Editor program.

4. Write your application programs using one of the Digital Research programming languages supported by Display Manager. Prepare the code using whatever Display Manager functions your program requires.

5. Compile your programs, and link to the necessary modules in the Display Manager Run-time Library.

Section 3: Terminal Setup Program

You can use DMSET, the terminal setup program, to tell Display Manager the characteristics of a terminal. You can also use it to create the Editor program for use with a design terminal, and to set up one or more run-time terminals for use with your application program.

The only requirements of a terminal used with Display Manager are that it have an addressable cursor, a clear screen command, and a minimum screen size of 24 rows by 52 columns. All other features are optional.

3.1 STARTING THE TERMINAL SETUP PROGRAM

To start DMSET, type the following command at your operating system prompt:

```
DMSET
```

The screen shows the Digital Research copyright banner, and a message asks you to wait while the terminal control codes from the TERMS.DM file are loaded into memory. The program then asks whether you want to run in Help or non-Help mode. Help mode provides detailed descriptions of each procedure before it is run; non-Help mode bypasses most of these descriptions, and permits the program to run faster. After you make this selection, the main Menu appears on your screen, as shown in the following figure.

```
                M A I N  M E N U

Option          Function
  E     Create EDITOR Program for Design Terminal
  W      WRITE Terminal Control Codes to disk file
  C     CUSTOM Terminal Setup
  T     TEST Terminal Control Codes
 ESC    Stop Terminal Setup Program (press ESC Key)

Please enter Your selection --> : :
```

Figure 3-1. Terminal Setup Program Main Menu

3.2 OVERVIEW OF TERMINAL SETUP PROGRAM OPTIONS

The next table provides brief explanations of the options available on this menu. Detailed explanations of these options occur later in this section, except for option C, which is explained in Appendix C.

Table 3-1. Terminal setup options

Format: Option

Explanation

Option E--Create Editor Program for Design Terminal
Before you can use the Editor program to design and create displays, you must create a version of it for a specific design terminal. Completing this option provides you with a version of the Editor tailored for use with a specific terminal.

Option W--Write Terminal Control Codes to Disk File
Use this option to create or extend a disk file containing the terminal control codes for the run-time terminals to be used with your application program. Your application program can later use the codes in this file to properly initialize the run-time terminal.

Option C--Custom Terminal Setup
Use this option to add, change, delete, or examine terminal control codes in the TERMS.DM file. Appendix C contains detailed instructions for custom terminal setups.

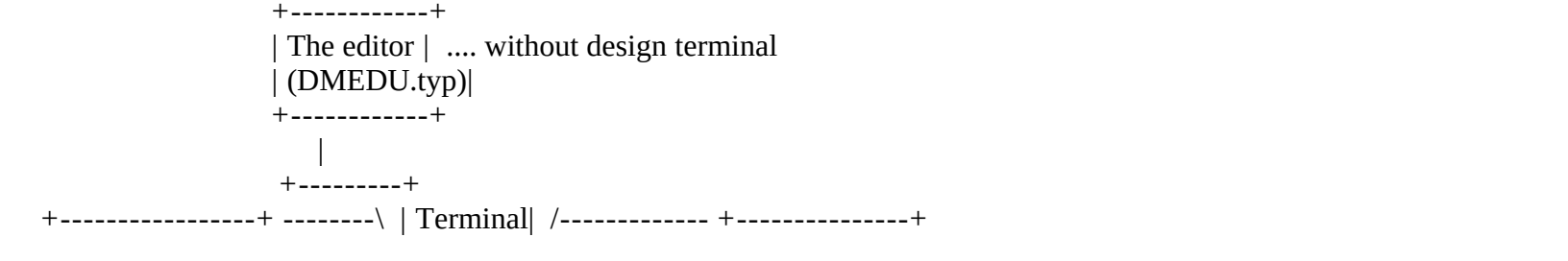
Option T--Test Terminal Control Codes
Use this option to verify that the control codes for a terminal in TERMS.DM are correct.

Option ESC--Stop Terminal Setup Program
Press the ESC key to stop the program and return control to your operating system.

3.2.1 OPTION E--CREATE EDITOR FOR DESIGN TERMINAL

DMEDU.typ is a program file found on your distribution disks. (The "typ" is the filetype used for program files in your operating system: COM for CP/M 2.2, CMD for CP/M-86, or EXE for MS-DOS.) This file contains a version of the Editor which is non-specific to any design terminal. When you create the Editor for your design terminal, you create a new version of DMEDU with a different name. You cannot use DMEDU as your Editor, or unpredictable results occur. Note that DMEDU, DMSET, and the TERMS.DM file must be on the same disk when creating the Editor.

The following figure "Create Editor for Design Terminal" illustrates the environment where you enter when creating the Editor for use with a design terminal. When you select option E from the Main Menu, a list of the terminals contained in the TERMS.DM file shows on your screen. You can scroll through this list to find the terminal that you want to set up as the design terminal for use with the Editor.



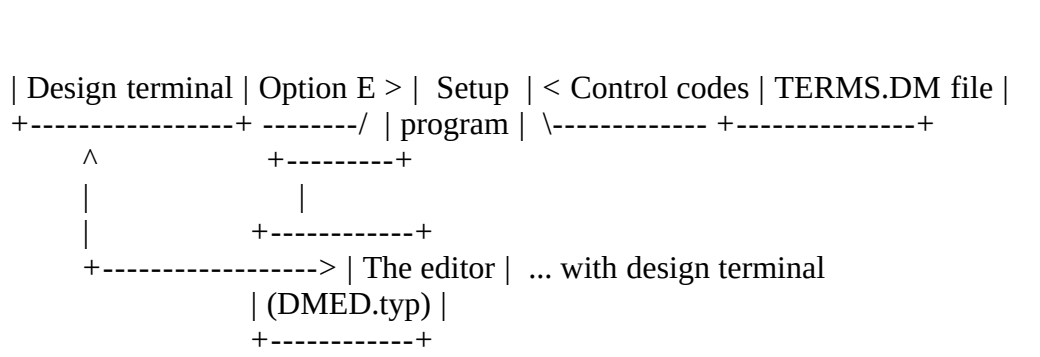


Figure 3-2. Create Editor for Design Terminal

When you see the design terminal on the list, you also see a 3-character code listed with it, such as A41 or Z11. Type this code to select the terminal from the list. If you type the code correctly, the word "FOUND" appears on the screen. Press RETURN, and terminal selection is complete.

If you type an incorrect code when selecting the terminal, the words "NOT FOUND" appear. Use the DEL key ("<--") to erase your entry, then enter a correct code. Note that you can also use the scroll commands Ctrl-W and Ctrl-Z, or the ESC key, at this time.

If you do not see your design terminal listed, you have 2 options as to how to proceed:

1. It might be that the design terminal uses the same control codes as another terminal already in the TERMS.DM file. If this is the case, select the terminal from the list, and then use the test option (option T on the Main Menu) to verify that the design terminal operates correctly.

2. If the design terminal is neither in the displayed list, nor uses the same codes as one that is, press ESC to return to the Main Menu. Then, select option C to do a custom terminal setup. See Appendix C for instructions.

After you select a terminal from the displayed list, your selection is displayed, along with its terminal control codes. The program then asks what name you want to assign to the version of the Editor that you are now creating. Your screen appears similar to the next figure.

```
The selected terminal is: <xxxxxxxx>
Terminal Control Codes: ABCD EFG3 MNPA BCDZ FGH9

Please select a name for the Editor. Press RETURN
to name the Editor DMED, type a different name,
or Press ESC to exit --> DMED

The Editor is now named DMED.typ
Press any Key to continue.
```

Figure 3-3. Terminal Control Codes/Editor Name Screen

Before proceeding, verify that you have selected the correct terminal. The name of the terminal that you selected replaces <xxxxxxxx>. If you selected

the wrong terminal, press ESC to make a different selection.

If you press RETURN, the Editor that you are creating is assigned the name DMED. The program automatically appends the appropriate filetype for your operating system, such as COM (for CP/M 2.2), CMD (for CP/M-86), or EXE (for MS-DOS).

To store the Editor on a drive other than your current one, precede the name with a drive specifier. For example, if you are logged to drive A but want to store the Editor on drive B under the name DMED, enter the following response:

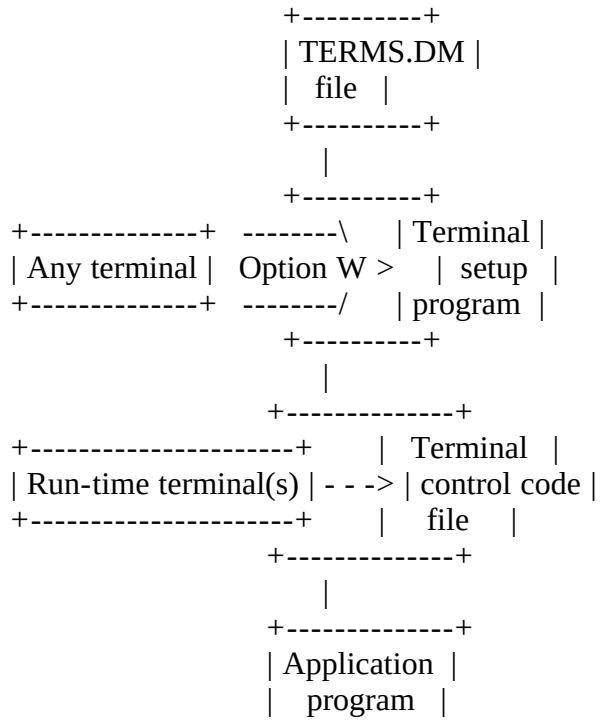
B:DMED

You can assign any valid filename to the Editor except DMEDU.typ, where "typ" is the filetype used for program files in your operating system. For example, DMEDU.COM is an unacceptable name in a CP/M-86 operating system environment.

After you have assigned a name for the Editor, press any key to return to the Main Menu. To run the Editor, exit from the DMSET program, and type the assigned name at your operating system prompt.

3.2.2 OPTION W--WRITE TERMINAL CONTROL CODES TO DISK FILE

This option allows you to write your application programs independent of the run-time terminals on which they are used. When you select this option, the terminals in the TERMS.DM file list on your screen. You can then select the terminals that you want to support in your application program from that list, and have them stored in a disk file that you name. Your application program can ask the end-user what run-time terminal is being used, and then read the correct control codes for that terminal from this file. The following figure "Write Terminal Control Codes to Disk File" illustrates this environment.



```
                    +--------------+
```

Figure 3-4. Write Terminal Control Codes to Disk File

After you select option W from the Main Menu, a prompt asks you to enter the name of the file to which you want to write the terminal control codes, as shown in the following figure.

```
WRITE TERMINAL CONTROL CODES TO DISK FILE

STEP 1-Indicate name of disk file to store code
STEP 2-Select codes to write in disk file

(Press ESC to exit this option. )
Enter file to write control code to:
```

Figure 3-5. Terminal Control Code Filename Prompt

The name that you enter must be an acceptable filename and filetype. If you enter the name of an existing file, you are asked if you want to add the codes for the terminals that you select to the end of that file, or overwrite it. If the file does not exist, you have the option to create it. Note that you cannot assign the name TERMS.DM to this file.

To create or access a file on a drive other than the one to which you are logged, precede the name with a valid drive specifier.

Next, you see the list of terminals in the TERMS.DM file. You can scroll through this list until you find the terminal that you want to add, then enter its 3-character code. If you enter a code that is not in the list, the words "NOT FOUND" appear. Use the DEL key ("<--") to erase the code, then enter a correct one. If you enter a code that the DMSET program cannot recognize, the words "BAD ENTRY" appear, and you must press ESC to re-enter the code.

When you enter an acceptable code, the word "FOUND" appears. Press RETURN, and you are shown which terminal you selected, along with its terminal control codes.

A series of prompts then ask you to do the following:

- Verify that you want to use the codes that you have selected as they are shown to you, one at a time.

- Indicate whether or not you want to select any more terminal control codes for your file.

- Confirm whether or not you want to write the selected codes into the file.

At the conclusion of this procedure, the program returns you to the Main Menu.

Note: Your Display Manager distribution disks contain a CBASIC Compiler program named INSTALL.BAS. This program is designed to install the end-user's terminal at run-time, using the control codes in the TERMS.DM file. This is an

alternate method to the one just described for installing the run-time terminal. You can modify INSTALL.BAS any way you want, and distribute it along with the TERMS.DM file and your application programs.

3.2.3 OPTION C--CUSTOM TERMINAL SETUP

The overview at the beginning of this section explains when you might need to use this option. Because this is not a commonly used option, it is discussed in Appendix C.

3.2.4 OPTION T--TEST TERMINAL CONTROL CODES

You can use this option to verify that the control codes for a design or run-time terminal are correct. You can only conduct this test using the actual terminal whose codes you want to verify. Testing the control codes for one terminal while using a different one gives unpredictable results, and might hang-up your terminal, forcing you to reboot your system.

When you select this option, the Terminal Test Menu appears on your screen, as shown in the following figure.

```
TERMINAL TEST MENU for <terminal name>

Option          Tests . . .
  A . . . . .  ALL features
  S . . . . .  Terminal STARTUP code
  P . . . . .  Cursor POSITIONING
  C . . . . .  CLEAR screen
  Z . . . . .  Screen SIZE
  O . . . . .  Cursor ON/OFF
  T . . . . .  STANDARD video attributes
  U . . . . .  USER-defined attributes
  M . . . . .  MULTIPLE attributes
  R . . . . .  Cursor ARROW KEYS
  F . . . . .  FUNCTION KEYS

Please enter Your selection --> : :
```

Figure 3-6. Terminal Test Menu

You conduct all tests interactively. Display Manager shows you a message explaining what results to expect, and asks if you want to continue. If you do, the test is performed, and the results show on your terminal screen. The program then asks you to confirm if the expected results happened. You then have the option to go on to the next test, or return to the Terminal Test Menu. Display Manager cannot determine whether or not the test was successful; you must decide on its success or failure, and react accordingly. If a particular test fails, it usually indicates that you need to change the control codes for the terminal that you are testing (see Appendix C).

Note that the Terminal Test Menu provides the option to test all features shown on the menu (option A), or only specific ones. If you select option A, the program conducts each of the other tests on the menu in turn. If you select an individual test, the program completes it, and then returns you to the Terminal Test Menu.

3.2.5 OPTION ESC--STOP TERMINAL SETUP PROGRAM

You can press ESC during any test, to terminate the testing procedures and return to the Main Menu.

Section 4: Display Design Concepts

A concept basic to understanding display design with the Editor is that designing the displays, and creating application programs to use those displays, are 2 separate steps. This section explains the fundamentals of display design. Section 5 describes the options found on the Editor Main Menu. Section 6 lists and discusses the Editor commmands. Section 7 explains how to use displays in your application programs.

The following figure "Editor Environment" illustrates the environment where you enter when using the Editor. In that environment, you design displays on the design terminal while running the Editor program (usually named DMED). When you complete the design of the display, you can store it in a file on one of your disks (a display file). Your application program subsequently reads the displays from the display file at run-time, and shows them on the run-time terminal when needed.

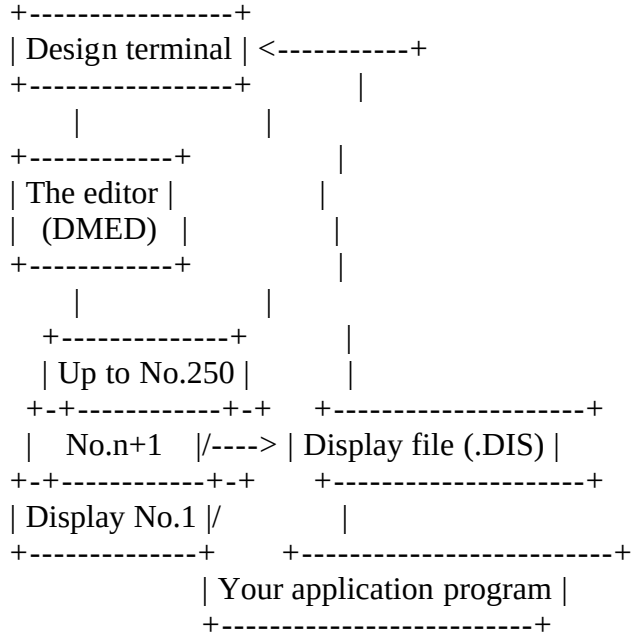


Figure 4-1. Editor Environment

You can also use the Editor to keep your display files accurate and up-to-

date. You can do the following:

- Recall displays from the display file to make corrections or changes.
- Delete a display from the file when it is no longer needed.
- Copy a display from a file, to use as a model when creating new displays.

Section 5 gives detailed explanations of these and other options.

4.1 DISPLAYS

A display is the information shown on a terminal screen. Displays usually cover the entire screen, but they can also cover only a portion of it. They serve to present information or instructions to the end-user, ask questions, present a list of options from which to choose, establish a form for data entry, or virtually any other purpose. The following figure is an example of a display showing a list of options for the end-user to choose. (Displays of this type are known as menus.)

```
        A C C O U N T S  P A Y A B L E

- - - - - - - - - - - - - - - - - - - - - - - - - - - -
                  MAIN MENU
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
Option                  Function
  1 . . . . .  Accounts Payable transaction maintenance
  2 . . . . .  Vendor maintenance
  3 . . . . .  Print Accounts Payable checks
  4 . . . . .  Print Accounts Payable reports
  X . . . . .  Stop Program/return to operating system

 : : < ----- Please enter your selection
```

Figure 4-2. Sample Menu Display

Each display in a display file receives a unique display reference number, ranging from 1 to 250, that you assign when creating the display. You can assign a new number at any time. Display reference numbers do not have to be contiguous.

Optionally, you can assign a display title in addition to the display reference number. The title bears no relationship to any other elements; it is strictly for your convenience in identifying one display from another when they list on your screen. Display titles can contain as many as 30 characters, including spaces. For example, "AP01/Accts Pay Main Menu" is an acceptable display title for the display shown in the preceding figure.

The Editor computes the screen size required to accommodate each display. The display size is based on the number of rows and columns required for the display. If your application program attempts to show a display on a screen with too few rows or columns, the display appears correctly only if all fields (including literal fields) are within the boundaries of the screen. Otherwise,

the results are unpredictable. This flexibility makes it possible to create displays for different sized screens.

Note: You cannot use the last column of the last row on the screen because, on many terminals, a character in this position causes the screen to scroll upward.

You can assign each display global values specifying whether or not to clear the run-time terminal screen before showing the display and, if your operating system supports the use of color, the color attributes to be applied. Section 5 discusses global values.

4.2 DISPLAY FILES

A display file contains the displays designed and saved using the Editor. You can store up to 250 displays in a display file. Your application program can use as many different display files as disk space on the run-time computer permits. However, only one display file can be open at any given time.

Assign your display files unique filenames that are compatible with your operating system. Filetypes can be used, and an informal standard of DIS is recommended. For example, a recommended name for an accounts payable display file might be ACCTSPAY.DIS.

4.3 DISPLAY FIELDS

Each display consists of one or more display fields. A display field is a portion of the display beginning at a particular row and column, and ending on the same row. The length of a display field can be from one column to an entire row, but fields cannot overlap.

Primarily, display fields do the following at run-time:

- Retrieve information entered by the end-user. These are input fields.

- Display variable information derived by your program from computations and data files. These are output fields.

- Display constant information, such as instructions, prompts, and field labels. These are literal fields.

The Editor assigns each input and output field in a display a field reference number when you create it. The Editor also provides an option to renumber one or more fields if needed. Field reference numbers range from 1 to 250.

You can assign video attributes to input and output fields. These attributes are the special effects that may be available on the run-time terminal, such as inverse video, full/half intensity, underlining, and color. If you assign a field a video or color attribute, but the feature is not available on the run-time terminal, the attribute is ignored.

4.3.1 Literal Fields

Literal fields contain information that is constant. For example, literal fields can serve as labels for input and output fields, column headings, data entry prompts, and instructions.

Literal fields cannot be changed during run-time. In fact, the end-user cannot move the cursor into a literal field.

Any field in a display not specifically defined as an input or output field is a literal field. Literal fields are not numbered, and cannot be assigned video attributes.

4.3.2 Input Fields

The end-user can enter information into input fields during run-time. Your application program can use the GETF (Get Field) or UPDF (Update Field) functions to retrieve information from input fields. Section 7 describes these and other functions.

Here are some of the ways you can control the input fields in your displays:

- Assign video or color attributes to the field.
- Assign initial (or default) values to the field.
- Place template characters in the field to aid the end-user during data entry.
- Retrieve information that the end-user enters into the field.
- Specify the type of data, such as alphabetic or numeric, that can be entered in the field.
- Specify what should happen if the end-user enters a special character into the field, such as an up-arrow or function key.
- Specify what should happen if the end-user enters an illegal character, such as the letter "a" in a numeric field.
- Display information in the field.

You can assign video or color attributes, or both, to an input field for special effects or increased visibility. The end of this section discusses these attributes.

When you define an input field with the Editor, you can enter an initial value in the field. When the display shows on the run-time terminal, the initial value appears in the field. Initial values can greatly simplify data entry for

an end-user, by showing the most commonly-entered value for a field. Consider the following example of a data entry prompt using an initial value:

```
Enter employee's hourly pay rate --> 12.00
```

The portion of the prompt "Enter employee's hourly pay rate -->" is a literal field. The input field begins in the column immediately following "-->" and has been assigned an initial value of 12.00. When the display appears on the run-time terminal, the prompt appears exactly as shown in the example. If the end-user presses RETURN without entering anything in the field, the value 12.00 is returned to the application program. The UPDF (Update Field) function retrieves information from a field containing an initial value.

Template characters are another way to simplify data entry for an end-user. Here is an example of a prompt using template characters:

```
Enter telephone number --> :(  )   -   :
```

The portion of the prompt "Enter telephone number -" and the two colons (:) are literal fields. The input field begins immediately following the first colon and ends at the column preceding the second colon. The parentheses, the space immediately following the parentheses, and the hyphen are all template characters. When the telephone number is entered, the cursor jumps over each template character. They cannot be typed over or erased. The input field in our example accommodates ten characters, for example, (206) 555-1212. Template characters are never returned to the application program.

You can combine template characters and initial values in an input field. Here is an example prompt:

```
Enter Social Security Number: nnn-nn-nnnn
```

The hyphens are template characters; the n's are the initial value of the field. This example of initial values differs from the preceding one in that, here, the value is not the most common response to the prompt; instead, each n serves as a place marker which you expect the end-user to replace with a number. As the n's indicate, the field accommodates nine characters, for example, 123-45-6789. The GETF (Get Field) function retrieves information from a field without returning its initial value.

You can define an input field in such a way that Display Manager validates each character as the end-user enters it at run-time. You can tell the Editor what type of data, such as alphabetic or numeric, you expect to receive in a field at run-time. Display Manager then ensures that only that type of data is returned to your program.

The Editor provides you with several ways to specify how the end-user must terminate data entry for a field. Data entry can be terminated when the field is full, an illegal character is entered (such as a number in an alphabetic field), an up or down cursor movement key is pressed, or a function key is pressed. The ENDF (End Field input) function can be called by your application program to determine precisely how the end-user terminated data entry. You can use the RESF (Resume Field entry) function to signal to the end-user that he entered an illegal character without terminating data entry for the field.

Your program can display information in an input field using the PUTF (Put data in Field) function. You can specify a format for the information when creating the field with the Editor. This is useful for assigning or changing a field's initial values.

4.3.3 Output Fields

You can use output fields to display variable information derived by your program. Use the PUTF (Put data in Field) function for this purpose.

Here are some of the ways you can control output fields in your displays:

- Display information in the field.
- Assign the field video or color attributes.
- Specify a format for the way information should appear in the field.
- Place template characters in the field.
- Assign the field initial values.

You can assign video or color attributes, or both, to an output field for special effects or increased visibility. This section explains these attributes in detail later.

When creating an output field with the Editor, you can specify a format for the way the information should appear. The format can specify that the information be left- or right-justified, a certain number of positions follow a decimal point, and more. Table 4-5, "Field Format Codes", later in this section explains the codes that determine the field format.

You can also place template characters and initial values in an output field. Template characters can enhance the appearance of an output field; initial values ensure the field's appearance when initially displayed.

Your program can retrieve information from an output field using the GETF (Get Field) or UPDF (Update Field) functions.

4.4 VIDEO ATTRIBUTES

You can assign video attributes to input and output fields to activate any special features available on the run-time terminal. You can assign the attributes when you create the field with the Editor. Of course, for the attribute to take effect, the specified feature must be available on the terminal; otherwise, Display Manager ignores the request.

Because video attributes cannot be assigned to literal fields, you might want to set up some of your prompts, help messages, and error description areas as input or output fields. For example, you might reserve the bottom row of your display for showing error messages. By making the row an output field, you can use video attributes to make the messages invisible until needed and highlighted when shown. Your application program can use the SETF function to

control video attributes during run-time.

Display Manager can accommodate up to 8 different video attributes for each display field. Each attribute has 2 settings, ON or OFF. You can assign a field 2 separate video attributes at the same time, provided that the terminal can handle combinations of features, and that the control codes for the terminal have been set up accordingly with the DMSET program (see Section 3 and Appendix C). If you assign multiple attributes but the run-time terminal cannot accommodate one or both, it ignores unsupported-attributes.

You cannot use simultaneous, multiple video attributes on some terminals, even though the attributes might be available individually. In such cases, Display Manager uses a priority scheme. The following table lists video attributes in their order of priority. Field visibility has the highest priority, followed by intensity, inverse video, flashing, underlining, and user-defined attributes one, two, and three.

You can assign these video attributes with the Editor. Section 7 explains which commands allow you to assign the attributes. In the following table, each attribute's normal state appears first; its special state shows second.

Table 4-1. Video attributes

Format: Attribute/Setting
    Feature/Effect

VISIBILITY
Determines if a field shows when initially displayed at run-time. It is useful for suppressing initial values, template characters, and so forth.

    Visible
    The field's contents appear on the screen.

    Invisible
    The field's contents are suppressed. The field can be made visible at run-time using the SETF (Set Field attributes) function.

INTENSITY
Determines how brightly the characters in the field are displayed.

    Full
    Characters in the field are displayed with full brightness.

    Half
    Characters in the field are displayed with 50% brightness. This setting is useful for prompts and help messages.

INVERSE VIDEO
Determines if characters are displayed as light images on a dark background or dark images on a light background.

    Normal
    Characters in the field are displayed as light images on a dark background.

Inverse
Characters are displayed as dark images on a light background.

FLASHING
Determines if characters in the field flash ON and OFF.

Normal
Characters in the field are displayed as constant images.

Flashing
Characters in the field blink ON and OFF continually. Flashing can attract attention to a field, but it annoys if overused.

UNDERLINING
Determines if the characters in the field are underlined.

Normal
Characters in the field are not underlined.

Underline
Characters in the field are underlined.

USER-DEFINED
Three user-defined attributes are available. They are used to activate Special features (such as color) that might be available on some terminals. The codes to activate these features must be included in the terminal's control codes (see Section 3).

Normal
The corresponding feature is not activated.

Special
The corresponding feature is activated for the field.

On some terminals, you must reserve a blank space on either side of a field in order for a video attribute to take effect. This technique of activating video attributes is the Plant method. Other terminals do not require blank spaces. They use the Paint method. Display Manager provides a function (RETDM) that can be called by your application program to determine which method (Plant or Paint) the run-time terminal uses.

If you are not sure whether the run-time terminals use the Plant or Paint method, you can always be safe by assuming the Plant method, and by reserving a space on either side of every input and output field in your displays.

4.5 COLOR ATTRIBUTES

You can assign distinct background and foreground colors to displays. You can also assign different background and foreground colors to the individual fields in a display. You have a choice of 8 different colors for backgrounds, and 16 different colors for foregrounds.

Of course, for color to take effect, the run-time terminal must have this feature. If the terminal is monochrome (black and white), Display Manager uses the video attributes in place of the color attributes. Consequently, it is a simple matter to design a display that works equally well on monochrome or color terminals. Your application program, at run-time, need not really be concerned with which type of terminal is in use.

While it is possible with Display Manager to design displays on a monochrome monitor for eventual use on a color monitor, do this with considerable care. This is also true when designing displays on one color monitor or video board for use on another. Colors vary significantly from one monitor to another, and do not always produce the anticipated result. A combination of colors that sounds good, or even looks good, on one monitor might wash out on another, rendering the information on the screen almost unreadable.

If you cannot test color combinations on the terminals where they will be used, use only those combinations of basic colors that provide high-contrast, or always use a black background.

Also be cautious of overusing colors in your displays. Reading a screen that is lit up like a Christmas tree can be both difficult and annoying.

The following table lists and explains the various color attributes that you can assign with Display Manager. The "Status Window" section explains how to set the attributes, and provides more detailed information.

Table 4-2. Color attributes

Format: Attribute
    Use and effect

Flashing
This can be set to cause the field to flash ON and OFF when the display is shown. Use with caution; a flashing field attracts attention, but can be overly distracting.

Background
While there are only 3 background color codes (RED, GREEN, and BLUE), they can be set in combinations to provide up to 8 different colors. In most cases, these 3 colors can be mixed as they would be on an artist's palette. For example, mixing RED and BLUE produces magenta.

Intensity
This applies only to the foreground color for the field. It can be set so that, when the foreground color is displayed, it is shown in bold intensity. This has the effect of producing a different color.

Foreground
The foreground color codes work in a manner identical to background codes, with one exception. When the intensity attribute is set, the result is a different color. For example, if red is selected as the foreground color and the intensity attribute is set, the actual color produced is light red. This doubles the number of available foreground colors, to 16.

4.6 STATUS WINDOW

Every input and output field in a display has a status window associated with it. You can make entries in the status window to assign various characteristics to each field. To gain access to a field's status window, place the cursor anywhere within the boundaries of the field, then press Ctrl-W. The status window appears on your screen with the cursor positioned inside.

Status windows for input and output fields are similar, but distinctly and logically different. The following figure shows a status window for an output field.

```
+-------------------------------------------------+
| Field No.  Row  Col  Len    Posts    Type-OUTPUT |
|   000    000  000  000     YY       *rr,cc*nnn  |
+-------------------------------------------------+
| Format  :L:  L,R,N,0-9,C,M     Comma   :N:   N,Y  |
|                                  |
| Video  :N:  :N:  :N:  :N:  :N:  :N:  :N:  :N:  N,Y |
|       Invs Half Invr Flsh Undl Usr1 Usr2 Usr3     |
| Color  :N:  :N:  :N:  :N:  :N:  :Y:  :Y:  :Y:  N,Y |
+--------fls--RED--GRN--BLU--Int--red--grn--blu------+
```

Figure 4-3. Output Field Status Window

The next figure shows a status window for an input field. Note that it contains everything that you find in an output field's status window (with one exception), plus 2 additional lines, the Validate/Beep line and the End Input line. The exception is that AutoRet replaces the Comma element.

```
+-------------------------------------------------+
| Field No.  Row  Col  Len    Posts    Type-INPUT  |
|   000    000  000  000     YY       *rr,cc*nnn  |
+-------------------------------------------------+
| Validate   :X: X,A,C,D,F,I,U     Beep    :N:   N,Y |
| Format     :L: L,R,N,0-9,C,M     AutoRet :N:   N,Y |
|                                  |
| End input---Cursor :N:   BadC :N:  FKey :N:    N,Y |
| Video  :N:  :N:  :N:  :N:  :N:  :N:  :N:  :N:  N,Y |
|       Invs Half Invr Flsh Undl Usr1 Usr2 Usr3     |
| Color  :N:  :N:  :N:  :N:  :N:  :Y:  :Y:  :Y:  N,Y |
+--------fls--RED--GRN--BLU--Int--red--grn--blu------+
```

Figure 4-4. Input Field Status Window

4.6.1 Status Window Elements

Because the elements within a status window are similar for input and output fields, the following discussion of these elements pertains to both windows.

When a distinction is necessary, [INPUT] or [OUTPUT], whichever applies, precedes the text.

Field No.
The number currently assigned to the field appears beneath the label. The Editor assigns every input and output field a unique number, ranging from 1 to 250, when it creates the field. You can use the renumber option (described in Section 6) to assign new numbers to one or more fields in your display. Note, however, that renumbering fields does not change their relative position within the display.

Row
This number indicates the row on your screen that contains the field. The top row on the screen is row 1.

Col
A number indicating the column of the first position in the field. The leftmost column on the screen is column 1.

Len
Indicates the number of positions (columns) in the field. Any template characters are included in the count.

Posts
The first letter indicates if the column immediately preceding the field contains a space (Y=Yes, N=No). The second letter indicates if a space immediately follows the field. These indicators are significant if any run-time terminal uses the Plant method to activate video attributes. In such cases, the values here should be YY.

If the run-time terminals use the Paint method, a space is not required on either side of the field, and the values here are unimportant.

Type-
Indicates whether the field is used for INPUT or OUTPUT.

rr,cc
These values indicate the row (rr) and column (cc) where the cursor is currently located. This can be different from the Row and Column numbers described earlier if the cursor is not in the first position of the field.

nnn
The value replacing nnn indicates the number of input and output fields in the display.

Validate :X: X,A,C,D,I,U,F
[INPUT] The code that you enter between the colons tells Display Manager how to validate information that the end-user types into this field at run-time. There are 7 different forms of validation from which to select, as explained in the following table.

Table 4-3. Input field validation codes

Format: Code

Validation type

X
Any printable character is accepted. This is the default for all input fields when they are first created (unless you assign other status window defaults using the Ctrl-UW command, see Section 6).

A
Only alphabetic characters and spaces are accepted. A numeric entry is treated as an illegal character.

C
Any characters, including control characters, are accepted, though function key input is not interpreted. Information entered by the end-user is not echoed back to the run-time terminal.

D
Only signed, decimal data is accepted. Alphabetic characters and more than one decimal point are treated as illegal characters. Spaces are allowed, but when the field contents are echoed, they are truncated starting at the first embedded or trailing space. Signs are moved next to the number, to eliminate embedded spaces.

I
Only signed, integer data is allowed. Alphabetic characters and decimal points are treated as illegal characters. Spaces and signs are handled as for code D.

U
Same as type X, except that all information entered by the end-user converts to upper case.

F
Only function keys are accepted as valid input for this field.

During run-time, Display Manager routines validate each keystroke as the end-user enters it. These routines do not pass unacceptable information to your application program.

You can select only one input data validation type code for each input field.

The next table shows what happens at run-time when different keys are pressed and a specific data validation code is in effect. Acceptable input is marked with Y; illegal or unacceptable input is marked N. Numbers refer to notes following the table.

Table 4-4. Interpretation of input validation types

Type of Data Entered by End-user	Validation Code used X	A	C	D	I	U	F
Alphabetic (A-Z, a-z)	Y	Y	Y	N	N	Y	N
Period	Y	N	Y	1	N	Y	N
Plus or Minus Sign	Y	N	Y	2	2	Y	N

Number (0-9)	Y	N	Y	Y	Y	Y	N
Space	Y	Y	Y	3	3	Y	N
Other printable character (1,",#,etc.)	Y	N	Y	N	N	Y	N
Control Key (Ctrl-A through Ctrl-Z) other than cursor movement	N	N	Y	N	N	N	N
Cursor movement (control keys and standard cursor arrow keys for editing)	Y	Y	4	Y	Y	Y	N
Function Key	5	5	5	5	5	5	Y

Notes:

1. A decimal field can have only one decimal point.

2. A single sign character can precede the digits in the field.

3. Spaces are allowed. When the field content is returned, it is truncated starting at the first embedded or trailing space. Signs are moved next to the number to eliminate embedded spaces.

4. The end-user cannot edit control fields because the control keys are stored as part of the field. The only way to terminate data entry in a control field is by pressing the RETURN key, or filling the field when AutoRet is appropriately set.

5. Function keys generate several, separate ASCII codes that appear to the computer as though several keys were pressed. Unless the FKey code is set, the generated characters are treated as though they were entered into the field individually. When FKey is set and the end-user presses a function key, Display Manager returns (via the ENDF function) only the appropriate negative value to indicate which function key. This assumes that the control codes passed via the INITDM function indicated that the terminal has function keys.

Beep :H:
[INPUT] If set to Y and the run-time terminal is equipped with an audible beeper, the beeper sounds when the end-user enters unacceptable information in the field. If N, the beeper does not sound under these same conditions.

Format :L: L,R,N,0-9,C,M
Specifies the way data should be formatted when placed in an input or output field. The next table describes the codes that you can use to specify formatting.

Table 4-5. Field format codes

Format: Code
Resulting Format

L
Left-justify. Aligns characters with the leftmost column of the field.

R
Right-justify. Aligns characters with the rightmost column of the field.

When a field's format code is L or R (signifying Left or Right justification, respectively) and truncation is necessary, information is always truncated from the right side of the data field.

N
Formats information in the field as pure, numeric data. Numbers are right-justified in the field, and leading zeros are removed. If, at run-time, the number is too long to fit in the field, the least significant digits to the right of the decimal point are truncated. If the number still does not fit, the field is filled with asterisks.

0-9
Formats information in the field as a decimal number. Enter a value from 1 to 9 to indicate the number of digits to the right of the decimal point. For example, if you enter 4, four digits follow the decimal point. Trailing zeros are inserted if the number contains less than four digits after the decimal point. If the number with the specified decimal places does not fit in the field, right truncation occurs, as in the "N" field. Leading zeros are removed and truncation, not rounding, eliminates extra decimal digits.

C
You can use this format code to send control keys to the screen. Information in the field is not formatted, and any number of characters can be sent to the field. You can use this command to go outside Display Manager and use special screen features, but use the command with caution. For example, this output format type is useful if you want to use a terminal feature that Display Manager does not support. To do this, create an output field with format code C, position the cursor in this field, send the control sequence for the terminal feature, and then make sure the cursor is back in this field before returning control to Display Manager.

M
Formats the field to contain money values. This automatically inserts a dollar sign or other currency symbol in the first space of the field, and formats it with 2 digits after the decimal point.

Comma :N:
[OUTPUT] This is the numerical separator code. If Y, a comma is inserted to the left of every 3rd digit to the left of the decimal point. For example, one million displays as 1,000,000.00. If N, commas are not inserted automatically.

AutoRet :N:
[INPUT] This code indicates whether or not data entry ends automatically when the input field is full. If Y, the end-user's data is returned to the program when a character is entered in the last position of the field or RETURN is pressed. If N, the end-user must press RETURN to terminate data entry.

Cursor :N:
[INPUT] This is the cursor arrow code. If Y, the up or down cursor arrow key, or the up/down cursor movement control key, causes data entry termination.

Note that cursor arrows might not be supported for a particular terminal. If N, the keys have no effect.

BadC :N:
[INPUT] This is the illegal character code. If Y, any illegal key entered forces data entry to end for the input field. An illegal key is any character that does not conform to the input format specified for the field (see Table 4-3, "Input Field Validation Codes"). Line editing keys are exempt from this check, unless the validation type is C (control field).

FKey :N:
[INPUT] This is the function key code. If Y, any supported function key that is entered causes data entry to terminate for this field. If N, any function key that is entered is not interpreted, and the characters sent when the key is pressed are treated as normal input for the field, unless the validation type is F (function key field).

For any given field, more than one code for ending data entry can be set at the same time. For example, setting the Cursor and FKey codes to Y terminates data entry if a cursor-positioning key or function key is entered. In this case, these keys provide the same functionality as the RETURN key. At run-time, the ENDF function can be used to find out how data entry was actually terminated (see Section 7).

4.6.2 Status Window Video Attributes

The next 8 field characteristics relate to the video attributes described earlier. Note that you can use the SETF function at run-time to override any of these settings. Also note that, with the exception of Invs, the run-time terminal must have the designated features available for these video attributes to take effect. Invs functions independent of terminal features.

:N: Invs
Set to Y to make the field invisible, so that it does not appear on the screen. The default, N, makes the field visible.

:N: Half
Set to Y to show the field in half intensity (50% brightness). The default, N, shows the field in full intensity.

:N: Invr
Set to Y to show the field in inverse video (dark images on a light background). The default, N, shows the field as light images on a dark background.

:N: Flsh
Set to Y to show the characters in the field flashing. The default, N, causes the characters not to flash.

:N: Undl
Set to Y to underline the characters in the field. The default, N, causes underlining not to take effect.

:N: Usr1
Set to Y to activate user-defined field attribute #1.

:N: Usr2
Set to Y to activate user-defined field attribute #2.

:N: Usr3
Set to Y to activate user-defined field attribute #3.

User-defined attributes can activate features available on the run-time terminal that are not otherwise supported by Display Manager. For example, if the terminal has a graphics mode capability, it can be defined as one of the user attributes. This requires that the control codes needed to activate this feature be included in TERMS.DM. (Section 3 and Appendix C explain how to do this.) If, for example, the graphics mode codes are set up as user-defined attribute number 1, and Usr1 is set to Y, the corresponding field is shown as a string of graphic symbols.

4.6.3 Status Window Color Attributes

The following are descriptions of the color attributes. Note that, with the exception of the "Invs" video attribute, color attributes always take precedence over the video attributes. Display Manager examines the control codes for the run-time terminal and, if color is available, uses the color attributes; otherwise, it uses the video attributes. Note that the resulting colors indicated might vary with different color graphics boards and terminals.

:N: Fls
Set to Y to cause the field to flash ON and OFF when the display is shown. A flashing field attracts attention but can also be distracting. Use cautiously.

:N: RED :N: GRN :N: BLU
These 3 codes specify the background color for the field. Any codes specified here take precedence over the global background color selected for the display. The codes are used in combinations to produce up to 8 different colors. The following table shows the colors normally produced by each combination.

Table 4-6. Background color codes

RED	GRN	BLU	Result
N	N	N	Black
N	N	Y	Blue
N	Y	N	Green
N	Y	Y	Cyan
Y	N	N	Red
Y	N	Y	Magenta
Y	Y	N	Brown
Y	Y	Y	white

The final 4 attributes serve together to specify the foreground color for the field.

:N: Int
When set to Y, the foreground color appears in full intensity. When N, it is shown in half intensity.

:Y: red :Y: grn :Y: blu
The foreground color codes function the same way as the background codes. However, when used in conjunction with the Intensity attribute (immediately preceding), up to 16 different colors are available. The next table shows the colors normally produced by each combination.

Table 4-7. Foreground color codes

Int	red	grn	blu	Result
N	N	N	N	Black
N	N	N	Y	Blue
N	N	Y	N	Green
N	N	Y	Y	Cyan
N	Y	N	N	Red
N	Y	N	Y	Magenta
N	Y	Y	N	Brown
N	Y	Y	Y	White
Y	N	N	N	Gray
Y	N	N	Y	Light Blue
Y	N	Y	N	Light Green
Y	N	Y	Y	Light Cyan
Y	Y	N	N	Light Red
Y	Y	N	Y	Light Magenta
Y	Y	Y	N	Yellow
Y	Y	Y	Y	Bright White

Section 5: Editor Options

You can do these things with the Editor:

- Create new displays
- Make changes to existing displays
- Delete obsolete displays from a display file
- Renumber the displays in a display file
- Change the currently open display file

The latter part of this section explains the Editor options in detail. However, before you can select an option from the Editor Main Menu, you must complete some preliminary steps to get the menu on your design terminal.

5.1 STARTING THE EDITOR

Unless the Editor was installed with a name other than the default, enter the following command at your operating system prompt to start the Editor running:

```
DMED
```

If the Editor was installed with a different name, enter that name (instead of DMED) to start the Editor.

Your screen then appears similar to the following figure.

```
-------------------------------------------------------
Display Manager 8x                    Version 1.0
Serial No. xxxx-0000-634321       All Rights Reserved
Copyright (c) 1983            Digital Research Inc.
-------------------------------------------------------

Display Manager installed for <xxxxxxxx>

(Press ESC to exit)
Press RETURN to continue
```

Figure 5-1. Editor Start-up Screen

Before moving on from this screen, check 2 important things. First, make sure that the copyright banner appears with your Display Manager version and serial number. If the banner does not appear, or has been modified, you might have an inoperable version of Display Manager. Please contact Digital Research immediately. Second, following the words "Editor installed for" (in place of <xxxxxxxx>) is the name of the design terminal for which the Editor is created. If this name does not match your terminal, your results are unpredictable. You should press ESC to stop the Editor and return to your operating system. See Section 3 for instructions about creating the Editor for use with your terminal.

When you are satisfied that you are using the correct Editor and terminal combination, press RETURN to continue. A prompt then asks you to enter the name of the display file that you want to use, as shown in the following figure.

```
Enter name of display file:
(Press RETURN to exit Display Manager)
```

Figure 5-2. Display File Name Prompt

Type the name of the display file you either want to create, or that contains the displays that you want to edit, or press RETURN to go back to your operating system.

The display file name can be qualified according to your operating system and hardware facilities. For example, enter B:ACCTSPAY.DIS to open a display file named ACCTSPAY.DIS on drive B while logged to drive A.

If you enter the name of a display file that does not currently exist on the

specified drive, an additional prompt appears on your screen, asking if this is the name of a display file that you want to create.

```
Enter Name of Display File:
(Press RETURN to exit Display Manager)

Do You want to create a new file (Y or N)? Y
```

Figure 5-3. New Display File Prompt

If you respond Y, the Editor creates the file you named. If you respond N, a prompt asks you to enter the name of a different display file. A message then appears to let you know that the Editor is busy opening the selected display file.

```
-> Opening display file . . .
```

Figure 5-4. Display File Open Message

Once the Editor Main Menu and a list of the displays in the file appear, you can begin editing your displays.

You can include the name of the display file that you want to use or create on the same line as the command to start the Editor. For example, you can type the following command at your operating system prompt:

```
DMED ACCTSPAY.DIS
```

The Editor checks whether the named display file exists (ACCTSPAY.DIS in this case). If it does not, you can create a file with that name, or enter a different name, as in the preceding examples.

5.2 EDITOR MAIN MENU

Once you have satisfactorily opened a display file, the Editor Main Menu appears on your screen:

```
    +--------------------------------------+
    |  D I S P L A Y   M A N A G E R       |
    |              by                      |
    |         DIGITAL RESEARCH INC.        |
    +--------------------------------------+

Current Display File:    A:ACCTSPAY.DIS
Current Display No:
Current Display Title:

                  M A I N  M E N U

              E -- Edit a Display
              D -- Delete a Display
              R -- Renumber the Displays
```

O -- Open a Display File
Q -- Help and Instructions
X -- Exit Display Monitor

Enter selection : :

Figure 5-5. Editor Main Menu

In addition to the available options, the Main Menu shows you information about the display file and the display currently being edited.

Current Display File:

The name of the display file that is currently open. A drive specifier precedes the name of the currently open display file. In the preceding example, the file ACCTSPAY.DIS is on the disk in drive A.

Current Display No:

Once you select a display to edit, its display reference number appears here.

Current Display Title:

If the display being edited was assigned a display title, it appears here.

The preceding example indicates a display file is open, but no display is currently being edited. The Main Menu appears this way until you select a display for editing.

The following table describes briefly the options on the Editor Main Menu. Subsequent parts of this section describe each option in detail.

Table 5-1. Editor main menu options

Format: Option
Function

E -- Edit a Display
Use this to create new displays, or to make changes to existing ones.

D -- Delete a Display
Use this to remove an obsolete display from a display file.

R -- Renumber Displays
Use this to change the display reference numbers assigned to the displays in a file.

O -- Open a Display File
Use this to close a display file and open a different (or the same) one.

Q (Quit) -- Help and Instructions
Use this to activate the on-line help facility. This facility provides detailed instructions on your design terminal screen for using the Editor.

X -- Exit Display Manager
Use this to stop the Editor, and to return control to your operating system.

5.3 OPTION E--EDIT A DISPLAY

Use this option to create a new display, or to copy or edit an existing display.

5.3.1 Creating New Displays

To create a new display, select the E option from the Editor Main Menu. The EDIT A DISPLAY screen then appears on your design terminal:

```
                EDIT A DISPLAY

    Current Display File:  A:ACCTSPAY.DIS
    Current Display No:
    Current Display Title:

    Enter DISPLAY REF NUMBER of Display to Edit : :
          (Press ESC to return to Main Menu)

    List of displays in file. (Scroll UP = ^W / DOWN = ^Z)
```

Figure 5-6. Edit A Display Screen (New Display File)

This figure indicates that the display file currently being used is on drive A, and named ACCTSPAY.DIS. As shown, no display from that file has been selected for editing; if it had been, the display reference number and title would also be shown. This screen also indicates that no displays are in ACCTSPAY.DIS; if there were, they would appear below the line.

A prompt asks you to enter the reference number of the display that you want to create. You can assign any number from 1 to 250. Type the reference number, and press RETURN.

Because there is no display in the file with the reference number that you assign, the Editor assumes that you want to create a new display, and shows you the NEW DISPLAY screen, as in the following figure.

```
                EDIT A DISPLAY

    Current Display File:  A:ACCTSPAY.DIS
    Current Display No:
    Current Display Title:

    NEW DISPLAY (Press ESCAPE to begin editing display)

    Enter Display Title --> :
```

Figure 5-7. New Display Title Screen

A prompt directs you to enter a title for the display that you are creating. The title can be as many as 30-characters long, and does not have to be unique within the display file. The title serves no other purpose than to help you identify it in the list shown on the EDIT A DISPLAY screen. This is the only opportunity you have to assign a display title.

After you enter the display title, this prompt appears on your screen:

Do you want to COPY an existing display? (Y=Yes/N=No) :N:

The option to copy an existing display is a useful method when you want to build displays from a prototype or a model. It is also the recommended technique when making changes or enhancements to existing displays, as it ensures against accidental destruction of the original version. You can find instructions for using this option later in this section.

5.3.2 Global Values

If you respond N (the default response) to the preceding prompt, the Global Values prompt appears, as shown in the following figure.

```
GLOBAL VALUES (Press ESC to begin editing display)

Clear screen? (Y=Yes, N=No) ?  :Y:  +-------- Color Menu --------+
                                    | 0 BLACK    8 Gray         |
Global Color Attributes             | 1 BLUE     9 Light Blue   |
-----------------------------------+ 2 GREEN    A Light Green  |
  Flashing? (Y=Yes,N=No)       :N: | 3 CYAN     B Light Cyan   |
  BACKGROUND COLOR (0-7)       :0: | 4 RED      C Light Red    |
  Foreground Color (0-9 or A-F) :7: | 5 MAGENTA  D Light Magenta |
                                    | 6 BROWN    E Yellow       |
                                    | 7 WHITE    F Bright White |
                                    +---------------------------+
```

Figure 5-8. Global Values Prompt

If you do not want to alter the Global Values for the display, press ESC.

Clear Screen Prompt

If you respond Y to the Clear Screen prompt, Display Manager clears the screen to all blanks before showing this display at run-time.

If you respond N, the display overlays whatever is on the screen from a previous operation. You can use screen overlays in several situations, such as to build screens line by line, keep help messages and instructions on the screen, and keep standard headings and banners intact.

Note: When one display overlays another, only the fields in the most recent display are accessible. Fields from any previous displays are not available to your application program.

Three separate global values can be set for color. Each of these values applies to the entire display, as opposed to individual fields within the display.

1) Flashing
   If set to Y, all literal fields in the display flash ON and OFF when shown. Use with caution.

2) BACKGROUND COLOR (0-7)
   You can specify any one of 8 different colors for the background of the display. Insert one of the codes (0 through 7) from the Color Menu shown to the right of the prompt.

3) Foreground Color (0-9 or A-F)
   Select any one of 16 different foreground colors for the display by entering the appropriate code from the Color Menu.

After you press ESC or reply to the global values prompts, the Editor clears the screen on your design terminal, and places the cursor in the top left corner. You can now create the actual display. Section 6 describes the various Editor commands that you can use to create the display. You can also enter Ctrl-QQ at this point, to display a list of the available commands on your terminal screen.

5.3.3 Copying Existing Displays

To create a new display by copying an existing one, reply Y when this prompt appears on the NEW DISPLAY screen:

```
Do you want to COPY an existing display? (Y=YES/N=No) :Y:
```

Figure 5-9. Copy Existing Display Prompt

When you indicate you want to copy an existing display, the following menu appears on the screen:

```
-------------------------------------------------------
      OPTIONS FOR COPYING AN EXISTING DISPLAY
-------------------------------------------------------

1 . . . . . Copy from CURRENT display file
2 . . . . . Copy from a DIFFERENT display file
3 . . . . . Quit Copy Options

Please enter Your selection --> : :
```

Figure 5-10. Copy Existing Display Options Menu

Use option 1 to copy the display from the currently open display file. The Editor then asks you for the reference number of the display to copy. If it finds the display, it copies it then places it on your design terminal for you to make changes. The display on your screen has the reference number you assigned to it when this session began.

If the display you want to copy does not exist, the Editor shows you an error message and asks you to enter another display reference number.

Use option 2 to copy the display from a file other than the one currently open. The Editor asks for the name of the file containing the display. Enter the file name, preceded by a disk drive specifier if necessary. If the Editor locates the file, it opens it, then the procedure is the same as if option one were selected. After the copy is made, the Editor closes the file from which the copy was made. You can now edit the copy.

If the Editor cannot locate the specified display file, an error message appears and asks you to re-enter the name correctly.

5.3.4 Editing Existing Displays

Once you have existing displays, you can select option E from the Editor Main Menu to change them. This places an EDIT A DISPLAY screen similar to the following figure on your screen:

```
                EDIT A DISPLAY

    Current Display File:  A:ACCTSPAY.DIS
    Current Display No:
    Current Display Title:

    Enter DISPLAY REF NUMBER of Display to Edit : :
    (Press ESC to return to Main Menu)

    List of displays in file. (Scroll UP = ^W / DOWN = ^Z)

    1    AP01/Accts Pay Main Menu  2    AP02/Vendor Maint Menu
    35   AP35/Vendor Maint Form    42   AP42/Print ChecKs
    103  AP103/Print Reports
```

Figure 5-11. Edit a Display Screen (Existing Display File)

Note that the display reference number and title of each display already in this file are listed below the line. If the file contains more displays than the screen can show at one time, use the Ctrl-W and Ctrl-Z commands to bring the others into view.

The prompt at mid-screen asks you to enter the Display Reference Number of the display you want to edit. If you enter the number of display that does not

exist, the Editor assumes you want to create a new display (explained earlier in this section). If the Editor locates the display you specify, it places it in memory and gives you the opportunity to change the global values for the display. Global values are also described earlier in this section.

Once you respond to the global values prompt, the Editor places the display you want to edit on your design terminal screen. Use the commands described in Section 6 to make changes to the display now on your screen.

5.4 OPTION D--DELETE A DISPLAY

To delete a display from the current display file, select option D from the Editor Main Menu. This causes the DELETE A DISPLAY screen to appear on your screen, as shown in the following figure.

```
                DELETE A DISPLAY

    Current Display File:  A:ACCTSPAY.DIS
    Current Display No:
    Current Display Title:

    Enter DISPLAY REF NUMBER of Display to Delete: : :
    (Press ESC to return to Main Menu)

    List of displays in file. (Scroll UP = ^W / DOWN = ^Z)

    1    AP01/Accts Pay Main Menu  2    AP02/Vendor Maint Menu
    35   AP35/Vendor Maint Form    42   AP42/Print ChecKs
    103  AP103/Print Reports
```

Figure 5-12. Delete a Display Screen

To delete a display from the current display file, enter its Display Reference Number. Once the display is deleted from the file, the program removes it from the list of displays shown on the screen.

If you enter a reference number for a display not in the file, an error message appears, asking you to enter a new number.

Note: Once you delete a display from the file, it is no longer available. If you do not have a back-up copy of the display, the only way to recreate it is as a new display.

5.5 OPTION R--RENUMBER THE DISPLAYS

Select option R from the Editor Main Menu to renumber individual displays or groups of displays in the display file. Note that renumbering displays does not change their relative position within the display file.

Note: Application programs use display reference numbers to place displays on

the run-time terminal at the appropriate time. Therefore, renumbering the displays can have significant impact on the operation of the program. The recommended practice is to make a copy of a display file currently being used before renumbering.

When you select the renumber option from the Main Menu, the RENUMBER DISPLAYS screen appears on your design terminal, as shown in the following figure.

```
                RENUMBER  DISPLAYS

    Renumber a group of displays or a single display.
    A single display has the same first and last number.

    Enter number of FIRST display in group    :   :
    Enter number of LAST  display in group    :   :

      Enter NEW starting reference number     :   :
      Enter INCREMENT value                   :   :

    (Press ESC to return to Main Menu)

    List of displays in file. (Scroll UP = ^W / DOWN = ^Z)

    1     AP01/Accts Pay Main Menu  2     AP02/Vendor Maint Menu
    35    AP35/Vendor Maint Form    42    AP42/Print ChecKs
    103   AP103/Print Reports
```

Figure 5-13. Renumber Displays Screen

5.5.1 Renumbering Groups of Displays

Enter the reference numbers of the first and last displays in the group to be renumbered. The last number must be greater than the first.

Then enter the new starting number for the group and the value by which you want the numbers to be incremented.

For example, assume you want to renumber displays 35, 42, and 103 shown in the preceding figure. The new reference numbers are to start at 20 and be assigned in increments of 10. Display 35 becomes number 20, 42 becomes number 30, and 103 becomes number 40. After the renumbering takes place, the new numbers are listed below the line. Here are the entries to make on the Renumber Displays screen and the resulting renumbered list:

```
                RENUMBER  DISPLAYS

    Renumber a group of displays or a single display.
    A single display has the same first and last number.

    Enter number of FIRST display in group    :35 :
    Enter number of LAST display in group     :103:
```

```
  Enter NEW starting reference number    :20 :
  Enter INCREMENT value                  :10 :

(Press ESC to return to Main Menu)

List of displays in file. (Scroll UP = ^W / DOWN = ^Z)

1    AP01/Accts Pay Main Menu  2    AP02/Vendor Maint Menu
20   AP35/Vendor Maint Form    30   AP42/Print ChecKs
40   AP103/Print Reports
```

Figure 5-14. Example of Renumbering Groups of Displays

5.5.2 Renumbering Individual Displays

To renumber a single display, enter the same reference number as the first and last in the group. Then, for the new starting reference number enter the number you want assigned to the display. Use any positive number for the increment value.

For example, assume you want to assign display number 103 in the "Renumber Displays Screen" the number 75. The following figure shows the entries to make on the Renumber Displays Screen and the resulting changes in the list below the line:

```
                RENUMBER  DISPLAYS

Renumber a group of displays or a single display.
A single display has the same first and last number.

Enter number of FIRST display in group    :03 :
Enter number of LAST display in group     :103:

  Enter NEW starting reference number    :75 :
  Enter INCREMENT value                  :1  :

(Press ESC to return to Main Menu)

List of displays in file. (Scroll UP = ^W / DOWN = ^Z)

1    AP01/Accts Pay Main Menu  2    AP02/Vendor Maint Menu
35   AP35/Vendor Maint Form    42   AP42/Print ChecKs
75   AP103/Print Reports
```

Figure 5-15. Example of Renumbering an Individual Display

Note: You cannot assign a different number to a display that would cause it to change its relative position in the display file. For example, you could not assign display number 2 as number 75 as this would require moving the display from its original position to between numbers 42 and 103.

5.6 OPTION O--OPEN A DISPLAY FILE

Select the O option from the Editor Main Menu when you want to close a currently opened display file. You can then either open a different file or reopen the same one.

This option has 2 primary purposes: to move between display files without stopping the Editor, and to periodically save your displays on disk. Display Manager works with the displays in a part of computer memory and only writes them to the disk file when necessary and practical. Closing the display file ensures the displays in memory are stored on disk. Recommended practice is to close the display file with this option at 10 to 15 minute intervals. This prevents accidental loss of your work from power or hardware failures.

After you select this option, the Editor displays the following message to let you know it is busy:

```
Now closing and Packing display file: A:ACCTSPAY.DIS
-> Please wait . . .
```

Figure 5-16. Display File Closing Message

The Editor packs a display file each time it is closed, to reclaim any wasted space and update the index. If the display file contains many displays, packing may take several seconds to complete, especially on a floppy disk.

After the display file is packed and saved, the Editor shows you the following prompt:

```
Enter Name of Display File:
(Press RETURN to exit Display Manager)
```

Figure 5-17. Open Another Display File Prompt

Enter the name of the display file you now want to open. If you enter the name of a file that is not on the current drive, the Editor asks if you want to create the file. You can create a new display file by replying Y, or enter a different name by replying N. A drive specifier can precede the file name.

Finally, the Editor opens the display file, and shows you the following message to let you know what is happening:

```
-> Opening display file ...
```

After the display file is opened, the program returns you to the Main Menu.

5.7 OPTION Q--HELP AND INSTRUCTIONS

The Editor provides an on-line Help facility. The facility is available in 2 forms, extended and limited. You can select the form you want by the way you configure your Display Manager disk.

1) Extended Help enables you to display complete descriptions of options on the Editor Main Menu and the Editor commands on your design terminal screen while using the Editor. It requires that the overlay file DMEDHLP.OVR be on the same disk as the Editor program.

2) Limited Help provides only a list of the Editor commands when you press Ctrl-OQ, Ctrl-Q?, or Ctrl-OU? while editing a display. To run the Editor with limited help, make sure the alternate overlay file DMEDOVR.OVR is on the disk, then remove or rename the overlay file DMEDHLP.OVR from your Display Manager disk.

Note that one of the above mentioned overlay files (DMEDHLP.OVR or DMEDOVR.OVR) must be on the same disk as the Editor program while it is running. If neither overlay is present, the Editor program cannot run. If DMEDHLP.OVR is present, Display Manager provides extended help; if DMEDHLP.OVR is not present, Display Manager provides only limited help.

You can activate the Help facility from any of 3 different places while using the Editor:

1) Editor Main Menu: Select the Q option.
2) Display Editing Screen: Enter Ctrl-QQ or Ctrl-Q?.
3) Output Options Menu: Select the ? option.

5.8 OPTION X--EXIT FROM THE EDITOR

To stop the Editor and return to your operating system prompt, select option X from the Main Menu. When you select this option, the Editor automatically closes the current display file before stopping. The following message appears to let you know the program is busy:

```
Now closing and Packing display file: A:ACCTSPAY.DIS
-> Number of times file has been edited: 25
   Display repacking timer --> 0
```

Figure 5-18. Editor Exit Screen

The message tells you the display file is being closed and packed. It also tells you how many times editing has been performed on the file. You might want to use this figure as an audit trail to determine how current your back-up files are, or if unauthorized changes have been made to the file.

The message also shows a "Display repacking timer -->". This timer counts down to zero simply as a way of indicating the status of the repacking process. When the timer reaches zero, packing is complete.

Section 6: Editor Commands

This section describes the various commands that you can use with the Editor

to create and make changes to your displays. You can use most of the commands in conjunction with the control key (labeled "Ctrl" on most keyboards). To enter these commands, hold down the Ctrl key while pressing the indicated letter key(s). For example, Ctrl-E means press the E key while holding down Ctrl.

The ESC key does the following:

- Ends certain commands, and returns you to the normal editing mode.

- Removes error messages, allowing you to continue processing.

- Functions as a general-purpose escape valve. If you do not know what to do next, press ESC several times to resume normal editing.

This section explains the Editor Commands by separating them into 4 categories, based on the type of function that the command performs:

1) Cursor Movement Commands
2) Field Editing Commands
3) Display Design Commands
4) Display File Commands

The following table lists the Editor commands by category. The remainder of this section provides a more detailed explanation of each command.

Table 6-1. Editor Commands by Category

Function	Command
CURSOR MOVEMENT COMMANDS	
Beginning of field	Ctrl-US
Beginning of next line	RETURN
Down half screen	Ctrl-QX
Down one line	Ctrl-X
End of field	Ctrl-UD
Left half screen	Ctrl-QS
Left one space	Ctrl-S
Next field	Ctrl-UF
Next word	Ctrl-F
Previous field	Ctrl-UA
Previous word	Ctrl-A
Right half screen	Ctrl-QD
Right one space	Ctrl-D
Tab	Ctrl-I
Up half screen	Ctrl-QE
Up one line	Ctrl-E
FIELD EDITING COMMANDS	
Boundary display (all fields)	Ctrl-QB
Boundary display (single line)	Ctrl-B
Change field to literal	Ctrl-UZ
Copy field to cursor location	Ctrl-UC
Define input field	Ctrl-UI

Define output field Ctrl-UO
Delete field Ctrl-UG
Move field right Ctrl-UV
Move field to cursor location Ctrl-UM
Renumber fields Ctrl-UR
Set Status Window as default Ctrl-UW
Status Window display Ctrl-W
Status Window display (constant)Ctrl-QW
Template insertion Ctrl-P
Video/Color attributes display Ctrl-QY
Video/Color attributes display Ctrl-Y

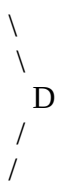
DISPLAY DESIGN COMMANDS
Center line Ctrl-OC
Delete character to left DEL ("<--")
Delete character under cursor Ctrl-G
Delete line Ctrl-QG
Delete word to right Ctrl-T
Draw border Ctrl-OB
Insert line Ctrl-QV
Insert space Ctrl-V
Print documentation Ctrl-OUW

DISPLAY FILE COMMANDS
Abandon without saving display Ctrl-OUQ
Change global values for display Ctrl-OUG
Save display, edit next one Ctrl-OUN
Save display, edit previous one Ctrl-OUP
Save display, edit same one Ctrl-OUS
Save display, return to Main Menu Ctrl-OUT
Write documentation Ctrl-OUW
Help instruction Ctrl-OU?

Note: Alternate commands exist for many of those shown in this table. For example, you can use Ctrl-UH instead of Ctrl-US to move the cursor to the beginning of a field. The individual command descriptions in this section show the alternate commands when available.

6.1 Cursor movement commands

The cursor movement commands are listed in the preceding table. You can use these commands to move the cursor around on the screen. Most of the commands center around what is known as the "Cursor Movement Diamond" on your keyboard. This diamond is formed by the relative position of the E, S, D, and X keys, as shown:



X

The relative position of the key in the diamond indicates the direction of cursor movement. Ctrl-E moves the cursor vertically up one line, Ctrl-S moves it one space to the left, Ctrl-D moves it one space to the right, and Ctrl-X moves it down one line.

If the cursor is in the far right column of a line and you press Ctrl-D, the cursor moves to the first column on the next line. Similarly, if the cursor is in the first column and you press Ctrl-S, the cursor moves to the last column of the previous line.

Note that the Editor does not allow you to position the cursor in the last column of the last row because, on many terminals, this causes an automatic scroll that disrupts the screen.

You can also control cursor movement with the standard ANSI cursor movement characters (Ctrl-K, Ctrl-H, Ctrl-L, and Ctrl-J), and by the cursor arrow keys if they generate the standard ASCII cursor codes shown previously. However, if the cursor arrow keys generate other control characters or escape sequences, the Editor might not be able to use these keys to move the cursor.

Beginning of Field: Ctrl-US or Ctrl-UH
Moves the cursor to the first column of the field. If the cursor is not in a field, an error message appears.

Beginning of Next Line: RETURN
Moves the cursor to the beginning of the next line in your display.

Down Half Screen: Ctrl-QX or Ctrl-QJ
If the cursor is in the top-half of the screen, it moves to the middle row. If it is in the bottom-half of the screen, it moves to the bottom row.

Down One Line: Ctrl-X or Ctrl-J
Moves the cursor downward to the next line in the display.

End of Field: Ctrl-UD or Ctrl-UL
Moves the cursor to the last column of the field. If the cursor is not in a field, an error message appears.

Left Half Screen: Ctrl-QS or Ctrl-QH
If the cursor is in the left-half of the screen, it moves to the leftmost column. If it is in right-half of the screen, it moves to the center column.

Left One Space: Ctrl-S or Ctrl-H
Moves the cursor to the left, one space at a time. If the cursor is in the leftmost column, it moves up to the end of the preceding line.

Next Field: Ctrl-UF
Moves the cursor to the next input or output field in the display. The cursor stops in the first column of the receiving field. The cursor moves from field-to-field in screen order, without regard to field numbers.

Next Word: Ctrl-F
Moves the cursor to the beginning of the next word. The next word is always to the right of the current word on the same row, or is the first word on the next row. The last character of the last word on a line is also treated as a "stop" point for this command.

Previous Field: Ctrl-UA
Moves the cursor to the previous input or output field in the display. The cursor stops in the first column of the receiving field. The cursor moves from field to field in screen order, without regard to field numbers.

Previous Word: Ctrl-A
Moves the cursor to the beginning of the previous word. The previous word is always to the left of the current word on the same row, or is the last word on the previous row. The last character of the last word on the previous line is also treated as a "stop" point for this command.

Right Half Screen: Ctrl-QD or Ctrl-QL
If the cursor is in the right-half of the screen, it moves to the rightmost column. If it is in the left-half of the screen, it moves to the center column.

Right One Space: Ctrl-D or Ctrl-L
Moves the cursor to the right, one space at a time. If the cursor is in the rightmost column, it moves down to the beginning of the next line.

Tab: Ctrl-I
Moves the cursor 8 columns to the right. The Editor does not use tabs in the normal way. Ctrl-I has the effect of pressing the SPACEBAR 8 times. For example, if you press Ctrl-I while defining an input or output field (Ctrl-UI or Ctrl-UO commands), the field is extended by 8 columns. On many terminals, Ctrl-I and the TAB key have the same effect.

Up Half Screen: Ctrl-QE or Ctrl-QK
If the cursor is in the top-half of the screen, it moves to the top row. If it is in the bottom-half of the screen, it moves to the middle row.

Up One Line: Ctrl-E or Ctrl-K
Moves the cursor up to the next line in the display.

6.2 FIELD EDITING COMMANDS

The field editing commands are listed in the table at the beginning of this section. Use these commands to establish the location, boundaries, and characteristics of input and output fields in a display.

To establish the location and boundaries of a field in your display, position the cursor where you want the field to begin, and press Ctrl-UI (for an input field) or Ctrl-UO (for an output field). Press the SPACEBAR to reserve each position in the field. Press ESC to end the field definition. Detailed descriptions of these and other commands follow.

Boundary Display (All Fields): Ctrl-QB
Displays the boundaries for all fields in the display. The next command description (Ctrl-B) explains field boundaries in detail.

While the boundaries are being displayed, you can move the cursor, and still use most editing commands. However, you cannot use some commands while boundaries are shown for the entire screen: attempts to use these commands result in error messages.

To remove the boundary display, press Ctrl-QB again.

Boundary Display (Fields on a Single Line): Ctrl-B
Displays the boundaries of all fields on the line containing the cursor. If there are no fields on the line, a message appears at the top of your screen. Field boundaries for output-fields are indicated by the letters O and o; the letters I and i mark input fields. You can identify adjacent fields because the letters shown alternate from upper- to lower-case, as in the following example:

IIiiii OOOOO oooo IIIIIIIII Code = 00

The example has seven fields. The first field is an input field two columns long (II). Adjacent to this field is another input field; it is 4-columns long (iiii). Next are 2 output fields and one input field, followed by the literal field "Code =" and an output field 2 columns in length.

Pressing any key makes the boundaries disappear.

Change Field to Literal: Ctrl-UZ
This command deletes a field, but leaves its initial value unchanged. You can use this command in conjunction with the Move Field to Cursor command, Ctrl-UM, to move literals around the display. To do this, make the literal an output field, move the output field with the Ctrl-UM command, then use Ctrl-UZ to delete the field, making the text once again a literal. Often, it is more efficient simply to retype the literal where you want it.

Copy Field to Cursor Location: Ctrl-UC
This command copies an input or output field to the current cursor position. Place the cursor in the row and column where you want the new field to begin, and press Ctrl-UC. The Editor asks you for the field number of the field to be copied. Enter the number, and the field is copied to the new location. The new field has the same attributes as the original field, and is assigned the next available field number. The command never overwrites an existing field in the display. In such instances, the Editor presents an error message, and the field does not copy.

Define Input Field: Ctrl-UI
This command reserves space in the display for an input field. To reserve the space, place the cursor in the row and column where you want the field to begin, and press Ctrl-UI. The Editor assigns the next available field reference number to this new field.

After you press Ctrl-UI, an @ symbol appears under the cursor, and all other

fields on the same row show their boundaries. Expand the field that you are creating by pressing the SPACEBAR. As each space is entered, another @ symbol reserves the column for this field (thus indicating the location and boundaries of the field).

Here is a recommended technique for creating and defining an input field into which you want the end-user to enter a social security number:

1. Place the cursor where you want the prompt for the field to begin, and type the prompt as a literal field. For example,

   Enter Soc. Sec. Number:

2. Move the cursor to where you want the input field to begin, and type the field's initial value (with template characters). The line might now look like this:

   Enter Soc. Sec. Number: nnn-nn-nnnn

3. Move the cursor back to the beginning of the field (to the first "n"), and press Ctrl-UI. An @ symbol replaces the n under the cursor.

4. Press the SPACEBAR 10 times to reserve space for the field. Your screen should now look like this:

   Enter Soc. Sec. Number: @@@@@@@@@@@

5. Press ESC, and the original, initial value that you typed for the field returns.

6. Move the cursor right 3 spaces to the first hyphen (-) and press Ctrl-P. Then, type the hyphen on your keyboard. Move the cursor 2 spaces to the right (to the next hyphen) and repeat the process. You have now defined the template characters for the field.

7. At this point, you can press Ctrl-W to display the field's status window.

To reserve 8 spaces for the field, press Ctrl-I. To delete a position from the field (thereby shortening it), press Ctrl-G. To add a position to the field, press Ctrl-V. Press ESC to signal the end of the input field; the boundaries for any other fields on the same line disappear.

When creating input or output fields, remember to reserve a space before and after the field if any run-time terminal uses the Plant method to activate video attributes. If the Plant method is used, do not use these spaces for another field or a literal. You can find out whether these spaces have been used by checking the Posts element in the field's status window. The first post indicates whether the character position immediately preceding the field is available to Display Manager at run-time for the Plant method of setting attributes. A Y means it is available; an N indicates it is not. The second post indicates whether the space immediately after the field is available. Your application program can also determine (at run-time) which method the terminal uses with the RETDM function.

If you attempt to create a field on top of an existing field, an error message appears, and the new field cannot be created.

Define Output Field: Ctrl-UO
This command reserves space in the display for an output field. The same rules, methods, and restrictions apply when creating output fields as when creating input fields. See the preceding command description.

Delete Field: Ctrl-UG
Deletes the field containing the cursor.

Move Field Right: Ctrl-UV
Moves the field containing the cursor one column to the right, and adds a space preceding the field. All fields to the right of the current field shift right one column. Any field positions pushed beyond the right edge of the screen are lost from the display.

Move Field to Cursor Location: Ctrl-UM
This command moves an input or output field to the current cursor location. Place the cursor in the row and column where you want the field to be located, and press Ctrl-UM. The Editor asks for the number of the field to be moved. Enter the number, and the field is moved. If the field that you are moving must overwrite an existing input or output field, an error message appears, and the move does not take place.

Renumber Fields: Ctrl-UR
Use this command to renumber all fields or a single field in the display. After you press Ctrl-UR, the Editor asks whether you want to renumber a single field or all fields. If you enter a number, that number is assigned to the current field, and all subsequently-created fields are numbered sequentially starting with the new number. If you respond with an A, all fields are renumbered in screen order sequentially from left to right, top to bottom, starting with field number one. Note that numbering rotates from 250 back to 1.

Press ESC to exit the renumbering mode.

Set Status Window Values as Default: Ctrl-UW
Sets the values in the current status window as the default for all subsequently-created fields. It is not necessary for the status window to be displayed for this command to work. Place the cursor in the field whose status window attributes are to serve as the defaults, then press Ctrl-UW.

As an example, assume that you want all output fields in your display to have a certain set of attributes. Here are the suggested steps:

1. Create the first output field, and assign the attributes for that field in its status window (use the Ctrl-W camand).

2. With the cursor in the output field, press Ctrl-UW to establish the current status window values as the defaults for all future fields.

3. Create the other output fields; they automatically have the same

attributes as the field created in the first step.

Status Window Display: Ctrl-W
Displays the status window for the field containing the cursor, and gives you the opportunity to review or change the attributes assigned for that field. After you press Ctrl-W, the status window appears with the cursor positioned in the first attribute. Use the cursor movement commands (Ctrl-S, Ctrl-D, Ctrl-E, or Ctrl-X), the RETURN key, or the SPACEBAR to move the cursor around inside the window.

If the cursor is in the top-half of the screen when you enter Ctrl-W, the status window appears in the bottom-half, and vice versa. If the cursor is not in a field when you press Ctrl-W, an error message appears.

After you have made all your changes in the status window, press Ctrl-W or ESC to resume editing.

Section 4 contains a complete description of the status window.

Status Window Display (Constant): Ctrl-QW
This command displays the status window, and leaves it on the screen while the cursor is moved around the display.

With this command, the cursor is not placed inside the status window. To make changes to the field's attributes, you must use the Ctrl-W command (as described earlier).

To remove the status window from your screen, press Ctrl-QW again.

Note: When the status window is ON, and the cursor is in an input or output field, the field's boundaries appear either immediately above or below it, and the remainder of the row showing the boundaries is set to blanks. Template characters appear as pluses (+) and minuses (-).

Template Insertion: Ctrl-P
Use this command to place a template character in the input field. You can use template characters to simplify data entry for the end user. See the Define Input Field command description for an example of how to insert template characters.

During run-time, the cursor does not come to rest on a template character, so the end-user cannot remove or alter them. Template characters are not returned to the application program.

To insert a template character in the field, move the cursor to the column where you want the character to appear, then press Ctrl-P. This reserves a space under the cursor for the character. Now, type the actual template character.

If the status window is being displayed, you see the field boundary for that column position marked with a plus sign (+). This means that the position contains a template character.

To remove a template character, place the cursor on the character, and press

Ctrl-P.

Video/Color Attributes Display: Ctrl-QY or Ctrl-Y
(Note that Ctrl-Y is inoperable with some operating systems.) Use these commands to see how your display would appear to the end-user during run-time. When either command is pressed, all color or video attributes assigned to the display, and available on the current design terminal, are shown. Note that color attributes take precedence over video attributes, and that you cannot edit while the attributes are being displayed. Use Ctrl-Y to display the attributes while the cursor is inside the status window.

Press any key to return to normal editing mode.

6.3 DISPLAY DESIGN COMMANDS

The display design commands are listed in the table at the beginning of this section. These are general-purpose commands to aid in the designing of displays.

Center Line: Ctrl-OC
Centers the line containing the cursor. All literals and fields shift appropriately to center the line. The command centers literals fields, initial field values, and template characters. If a field has no initial value or template characters, it is not centered.

Delete Character to Left: DEL ("<--")
Deletes the character immediately to the left of the cursor. All characters to the right on the same row are shifted left one space. If you remove all positions from a field with the DEL key ("<--"), that field is deleted from the display.

Typing over characters already on the screen replaces those characters with the typed ones.

Delete Character Under Cursor: Ctrl-G
Deletes the character under the cursor. All characters to the right on the same row shift left one space. If you remove all positions from a field using Ctrl-G, the field is deleted from the display.

If border-mode is ON (Ctrl-B command), Ctrl-G reduces the size of the field containing the cursor, without affecting the initial value of the field (if present).

Delete Line: Ctrl-QG
Deletes the line containing the cursor. The line is removed from the display, and all lines below the cursor move up one row.

Delete Word to Right: Ctrl-T
The word or string of spaces immediately to the right of the cursor position is removed from the display. All remaining characters to the right of the cursor on the same row are shifted left to fill the vacated spaces.

Draw Border: Ctrl-OB
This command provides an easy way to draw borders or other figures in your displays. Press Ctrl-OB and the Editor asks for the character you want to use for drawing. Press this character, and then proceed to draw using the cursor movement commands.

To change to another character for drawing, just type the new character while border-mode is ON. To end border-mode, press ESC.

Insert Line: Ctrl-QV
Inserts a Carriage Return at the current cursor position, creating a blank line. To use this command, put the cursor at the space where you want the new line, and press Ctrl-QV. The line to the right of the cursor moves down one row, and the newly-inserted row to the right of the cursor is blank. The last row on the screen is pushed off the screen, and is lost.

Insert Space: Ctrl-V
Inserts a character between 2 existing characters in a literal field, or expands an input or output field. To use this command, place the cursor where you want the space inserted, and press Ctrl-V. The Editor inserts a blank space under the cursor. The characters to the right of the cursor, and on the same row (including the one previously under the cursor), shift one space to the right. Characters on other lines are not affected. Literals or field positions pushed beyond the right margin of the screen are lost.

If boundary mode is ON (Ctrl-QB command), Ctrl-V expands the size of the field containing the cursor, without affecting the initial value of the field (if present).

Print Documentation: Ctrl-OUW
Prints documentation for the current display or, alternately, places it in a separate ASCII-type disk file for subsequent processing.

The resulting documentation accounts for every field in the display, whether visible or invisible. It documents literal fields and template characters, as well.

After you press Ctrl-OUW, the following message appears to explain your options:

```
Hit P for document to printer
Hit D for add to end of file (DISPLAY.DOC)
Output Option: Enter letter or ESC -->
```

Figure 6-1. Documentation Options Menu

If you select option D, the Editor adds the documentation to the end of a disk file named DISPLAY.DOC, if the file exists. If it does not exist, the Editor creates it on the same disk containing the current display file. DISPLAY.DOC is an ASCII file that can be edited with a word processor or other program that reads standard ASCII files. Each line is terminated with a Carriage Return/Line Feed sequence, and the end-of-file is marked with the standard Ctrl-Z character. This facility is provided as an aid for creating user manuals and program documentation. It is also an efficient method, when

documenting several displays, for printing at a later time.

Press ESC at any time to abandon the documentation process.

The next 2 figures illustrate the documentation created by pressing Ctrl-OUW. The first figure is a sample display. The second is a partial example of the documentation created by the Editor for that display. (An image of the display is also produced as part of the documentation.) Note in the first figure that fields have been numbered to correspond to those in the second (this is not a normal function of the documentation process). Also note in the second figure that fields are listed by row and column. All information found in the field's status window is shown. If assigned, initial values are shown just below the field number.

Several fields in the sample display have the INVISIBLE attribute turned ON. These fields do not appear when the display is initially shown; they must be activated using the SETF function in the application program. Field 75, for example, is made visible only after the end-user enters all required customer information and begins to enter information in the QTY field. The initial value in field 75 tells the end-user to complete the CUSTOMER PHONE ORDER by setting the QTY field to zero.

```
                CUSTOMER PHONE ORDER

 ---------------------------------------------------------------------
| Customer: _____________________  Address: _____________________   |
| City: _____________________ State: __ Zip: _____ Phone: (___) ___-____   |
| Payment: _  A = Account, B = Bank Card, C = C.O.D.   No. _____________   |
 ---------------------------------------------------------------------
 Press ESC to exit. Set QTY = 0 to send order.
 Press CTRL Z for Part No. Reference
 ------------------------------------------------------------
| QTY |    DESCRIPTION    | PART NO. | PRICE EA ||  TOTAL   |
 ============================================================
|     |                   |          | $      ||         |
|     |                   |          | $      ||         |
|     |                   |          | $      ||         |
|     |                   |          | $      ||         |
|     |                   |          | $      ||         |
 ------------------------------------------------------------
                          TOTAL SALE ||         |
                                     -----------
 INCORRECT PART NO. ENTERED (PRESS CTRL Z FOR HELP)
 MUST BE A, B, OR C
 ORDER SENT, PRESS RETURN TO CONTINUE
```

Figure 6-2. Sample Display

```
                V B F A C C B F I H I F U U U U
                A E O U O U A . N A N L N S S S
                L E R T M R D K V L V A D E E E
                I P M . M S K E I F E S L R R R
Num Typ Row Col Len Post D . T R A R Y Y S . R H N 1 2 3
```

```
--- --- --- --- --- ---- ------------ ------- ----------------------
50 IN   4   13  21  YY  X  N  L  Y    N  Y  N  Y  N  N  N  N  N  N  N
_____________________
--- --- --- --- --- ---- ------------ ------- ----------------------
51 IN   4   45  23  YY  X  N  L  Y    N  Y  N  Y  N  N  N  N  N  N  N
_____________________
--- --- --- --- --- ---- ------------ ------- ----------------------
52 IN   5   9   16  YY  X  N  L  Y    N  Y  N  Y  N  N  N  N  N  N  N
_______________
--- --- --- --- --- ---- ------------ ------- ----------------------
54 IN   5   33  2   YY  U  N  L  Y    N  Y  N  Y  N  N  N  N  N  N  N
__
--- --- --- --- --- ---- ------------ ------- ----------------------
55 IN   5   41  5   YY  I  N  L  Y    N  Y  N  Y  N  N  N  N  N  N  N
____
--- --- --- --- --- ---- ------------ ------- ----------------------
56 IN   5   54  14  YY  I  N  L  Y    N  Y  N  Y  N  N  N  N  N  N  N
(___) ___-____
--- --- --- --- --- ---- ------------ ------- ----------------------
57 IN   6   12  1   YN  U  N  L  Y    N  Y  N  Y  N  N  N  N  N  N  N
_
--- --- --- --- --- ---- ------------ ------- ----------------------
103 IN  6   54  14  YY  I  N  L  Y    N  Y  N  Y  N  N  N  N  N  N  N
No. ___________
--- --- --- --- --- ---- ------------ ------- ----------------------
77  OUT 8   2   18  YN      L    N        N  Y  N  N  N  N  N  N
Press ESC to exit.
--- --- --- --- --- ---- ------------ ------- ----------------------
75  OUT 8   21  26  NY      L    N        Y  Y  N  N  N  N  N  N
Set QTY = 0 to send order.
--- --- --- --- --- ---- ------------ ------- ----------------------
76  OUT 9   2   35  YY      L    N        Y  Y  N  N  N  N  N  N
Press CTRL Z for Part No. Reference
--- --- --- --- --- ---- ------------ ------- ----------------------
1   IN  13  3   3   YY  I  N  L  Y    N  Y  N  N  N  N  N  N  N  N

--- --- --- --- --- ---- ------------ ------- ----------------------
2   IN  13  9   17  YY  X  N  L  Y    N  Y  N  N  N  N  N  N  N  N

--- --- --- --- --- ---- ------------ ------- ----------------------
4   IN  13  30  5   YY  I  N  L  Y    N  Y  N  N  N  N  N  N  N  N

--- --- --- --- --- ---- ------------ ------- ----------------------
19  IN  13  41  7   YY  D  N  R  Y    N  N  N  Y  N  N  N  N  N  N  N

--- --- --- --- --- ---- ------------ ------- ----------------------
24  OUT 13  51  9   YN      M    N        N  N  N  N  N  N  N  N
```

Figure 6-3. Sample of Display Documentation

6.4 DISPLAY FILE COMMANDS

The display file commands are listed in the table at the beginning of this section. Use these commands to save a display that you have been creating or changing with the Editor. All of these commands have a prefix of OU. When you press Ctrl-OU, the Editor shows you the following menu:

```
                Enter letter or ESC -->

   G - Global Changes          N - Save & Edit Next
   P - Save & Edit Prior       O - Abandon, No save
   S - Save & Edit Same        T - Save & Main Menu
   W - Write Documentation     ? - HELP Instructions
```

Figure 6-4. Output Options Menu

Press ESC to resume editing the current display.

Options N, P, S, and T save whatever work you have done on the display in an area of memory called a buffer. These options do not necessarily cause the changes to be stored in the display file on disk. To ensure that your displays are stored on disk, select the T option here to return to the Main Menu. Select the O option at that menu, which closes the display file. Closing the file flushes everything from the buffer area, and writes it to the disk. You should do this periodically, to make sure that your work is not lost in the event of hardware or power failures.

Abandon Work, Do Not Save Display: Ctrl-OUQ
Abandons editing of the current display, and loses any changes made since the last time the file was saved using one of the options on the Output Options Menu. When you select this option, the Editor asks you to confirm your choice before actually abandoning your work. If you do confirm, you are returned to the Main Menu; otherwise, editing resumes.

Change Global Values: Ctrl-OUG
This option provides you the opportunity to change the current global values for the display. "Global Values" in Section 5 describes the changes that you can make.

Save Display, Edit the Next One: Ctrl-OUN
Saves the current display, locates the next one in the display file (according to the display reference numbers), and places it on your design terminal screen for you to edit.

If the current display is the last one in the file, an error message appears, and you are returned to the Main Menu.

Save Display, Edit the Previous One: Ctrl-OUP
Saves the current display, locates the previous one in the display file (according to the display reference numbers), and places it on your design terminal screen for you to edit.

If there is no previous display in the file, an error message appears, and you are returned to the Main Menu.

Save Display, Resume Editing Same One: Ctrl-OUS

Saves the current display, then makes it available again for editing.

Save Display, Return to Main Menu: Ctrl-OUT
Saves the current display, and returns you to the Main Menu.

Write Documentation: Ctrl-OUW
This option is explained under "Display Design Commands" earlier in this section.

Help Instructions: Ctrl-OU?
This option displays a list of the Editor commands on your design terminal screen.

6.5 EDITOR COMMANDS SUMMARY

This section summarizes the Editor commands by listing them alphabetically by commands. Use the following to interpret the Category column:

CM Cursor Movement
DD Display Design
DF Display File
FE Field Editing

Table 6-2. Editor Commands Summary

Command	Category	Description
Ctrl-A	CM	Cursor to previous word
Ctrl-B	FE	Boundary display (single line)
Ctrl-D	CM	Right one space
Ctrl-E	CM	Up one line
Ctrl-F	CM	Right to next word
Ctrl-G	DD	Delete character under cursor
Ctrl-H (*)	CM	Left one space
Ctrl-I	CM	Moves to the next tab stop. If used while defining a field, reserves 8 spaces.
Ctrl-J (*)	DM	Down one line
Ctrl-K (*)	CM	Up one line
Ctrl-L (*)	CM	Right one space
Ctrl-P	FE	Insert template character
Ctrl-S	DD	Left one space
Ctrl-T	DD	Delete word to right
Ctrl-V	DD	Insert space
Ctrl-W	FE	Display/Change status window
Ctrl-X	CM	Down one line
Ctrl-Y	FE	Show video/color attributes
Ctrl-OB	DD	Draw border
Ctrl-OC	DD	Center line
Ctrl-OUG	DF	Change global values
Ctrl-OUN	DF	Save display-edit next one

Ctrl-OUP	DF	Save display-edit previous one
Ctrl-OUQ	DF	Abandon without saving display
Ctrl-OUS	DF	Save display-edit same one
Ctrl-OUT	DF	Save display-go to Main Menu
Ctrl-OUW	DD	Write display documentation
Ctrl-OU?	DF	Display Editor commands list

Ctrl-QB	FE	Boundary display (all fields)
Ctrl-QD	CM	Right half screen
Ctrl-QE	CM	Up half screen
Ctrl-QG	DD	Delete line
Ctrl-QH (*)	CM	Left half screen
Ctrl-QJ (*)	CM	Down half screen
Ctrl-QK (*)	CM	Up half screen
Ctrl-QL (*)	CM	Right half screen
Ctrl-QS	CM	Left half screen
Ctrl-QV	DD	Insert Line
Ctrl-OW	FE	Leave status window ON
Ctrl-QX	CM	Down half screen
Ctrl-QY	FE	Video attributes display

Ctrl-UA	CM	Previous field
Ctrl-UC	FE	Copy field to cursor
Ctrl-UD	CM	End of field
Ctrl-UF	CM	Next input/output field
Ctrl-UG	FE	Delete field
Ctrl-UH (*)	CM	Beginning of field
Ctrl-UI	FE	Define input field
Ctrl-UL (*)	CM	End of field
Ctrl-UM	FE	Move field to cursor
Ctrl-UO	FE	Define output field
Ctrl-UR	FE	Renumber fields
Ctrl-US	CM	Beginning of field
Ctrl-UV	FE	Move field right
Ctrl-UW	FE	Set status window as default
Ctrl-UZ	FE	Change field to literal

DEL key	DD	Delete character to left
ESC key		Terminates some commands
RETURN	CM	Beginning of next line down
TAB key	CM	Moves to next tab stop. If used while defining a field, reserves 8 spaces.

(* = Alternate command. See Section 6-1, "Cursor movement commands".)

Section 7: Applications Programming

This section explains how to design and code application programs to use displays designed with the Editor.

7.1 OVERVIEW OF APPLICATIONS PROGRAMMING

The Display Manager Run-time Library contains the functions that you need to manage displays created with the Editor. When you write your application program source code using one of the Digital Research programming languages, you code in the necessary calls to these functions. Your source program is then compiled using the appropriate Digital Research compiler, thus producing a program object module. The object module is linked to the Run-time Library, and the resulting component is distributed for use on the end-user's run-time computer.

Your "Display Manager Programmer's Guide" contains instructions for compiling and linking your application program source code. The remainder of this section provides instructions for constructing calls to the various Display Manager functions in your source code.

The following figure "Application Programming Environment" illustrates the applications programming environment.

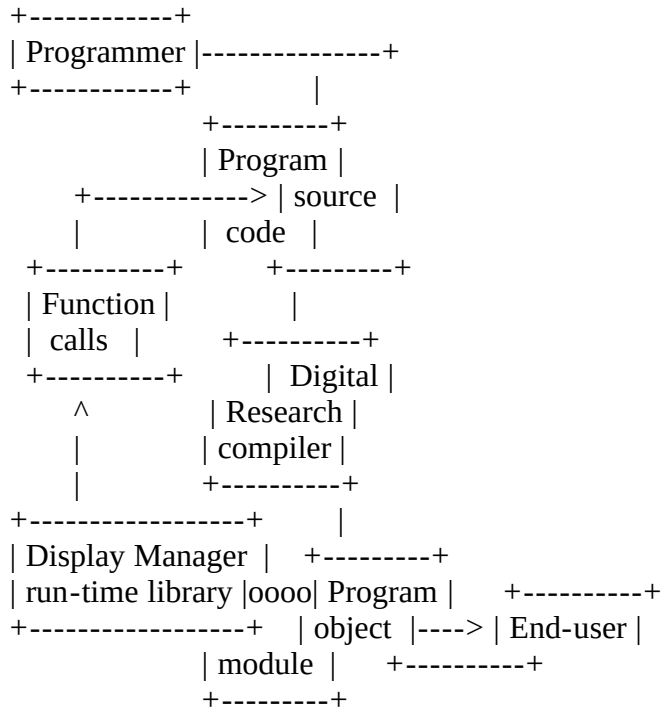


Figure 7-1. Application Programming Environment

7.2 FUNCTION CATEGORIES

Display Manager has 3 categories of functions. These categories serve only to aid your understanding of the functions, not to restrict their use.

1) The Initialization Category consists of only one function. Use it to initialize your application program and the run-time terminal for use with Display Manager.

2) The Display Management Category contains functions that find out what

features are available on the run-time terminal, open and close display files, clear the terminal screen, show a display on the screen, and (if the terminal has the capability) turn the cursor ON or OFF.

3) The Field Management Category has functions that you can use to determine the characteristics of a field in a display, retrieve and validate information entered into a field by an end-user, determine how the end-user stopped entering information into a field, and much more.

The following table lists the functions by category, and shows the mnemonic name assigned to each. The remainder of this section describes each function in detail. The descriptions are alphabetized according to the mnemonic function name.

Table 7-1. Display Manager Functions by Category

FUNCTION Description	Mnemonic
INITIALIZATION	
Initialize run-time terminal and program	INITDM
DISPLAY MANAGEMENT	
Clear screen	CLRSCR
Close display file	CLSDIS
Open display file	OPNDIS
Place display on screen	DISPD
Return run-time terminal attributes	RETDM
Set cursor visible/invisible	CURS
FIELD MANAGEMENT	
Determine data entry termination method	ENDF
Determine field position, length, type	RETF
Display data in field	PUTF
Modify field attributes	SETF
Place cursor in specific field	POSF
Place cursor in relative field	NXTF
Resume data entry	RESF
Retrieve/validate field input	GETF
Retrieve/validate field input/initial value	UPDF

7.3 FUNCTION DESCRIPTIONS

This section contains detailed descriptions of the Display Manager functions, and explains how to use them in your application programs. The following information about each function is provided when appropriate:

"Syntax" explains the way the various elements in the function call must be arranged when coded in your application program language. Pascal users please note that function calls are assignment statements, and not logical compares.

"Function Category" indicates the general category to which the function belongs, and is shown separately.

"Explanation" provides a brief description of what the function is designed to do.

"Cursor Disposition" indicates the location of the cursor following completion of the function. If omitted, the cursor's location is not affected.

"Function Argument" explains what values must be passed in the function argument as part of the function call. Some functions do not pass an argument.

"Return Values" describes the information returned by the function at its conclusion. It is a recommended programming practice to thoroughly check the value returned by the function, since this value indicates whether or not the function was successful.

"Additional Comments" explains any special situations and restrictions that you must observe when using the function.

"Example" is a segment of program code showing one way to use the function in an application program. The examples are in CBASIC Compiler, but generalized so they can be applied to any other high-level programming language. In these examples, variable names ending with a dollar sign ($) denote variable length, string data. Names ending with a percent sign (%) denote fixed length, binary integer data. Line numbers have been inserted for reference, and are not part of the actual code. Note that the examples call the DM.ERR routine following each function. This routine checks the returned value, to determine if the function completed successfully.

CLRSCR Function: Clear Screen

Syntax: <integer variable> = CLRSCR

Display Management Function

Explanation:
The CLRSCR function clears the run-time terminal screen by placing blanks in all positions.

Cursor Disposition:
The cursor is placed in the top-left corner of the screen.

Return Values:
The CLRSCR function always returns a zero.

Additional Comments:
While designing a display with the Editor, you can specify that the screen must always be cleared before the display is shown. In such cases, using the CLRSCR function is unnecessary. However, if you did not specify that the screen be cleared but now find it necessary, the CLRSCR function does this for you.

Example:
The run-time terminal screen is cleared in line 300. Subsequent lines close the display file, check for errors during the close function, and stop the program.

```
300 DONE: RET.ERR% = CLRSCR          REM Clear terminal screen
310       RET.ERR% = CLSDIS          REM Close the display file
320       CALL DM.ERR(RET.ERR%,CLSDIS$) REM Check for file error
330       STOP                       REM Stop the program
```

CLSDIS Function: Close Display File

Syntax: <integer variable> = CLSDIS

Display Management Function

Explanation:
Closes the currently open display file.

Return Values:
Zero is returned if the file is closed successfully; otherwise, a negative value is returned.

Additional Comments:
The recommended programming practice is always to close the current display file before terminating your application program.

Display Manager transmits the terminal control codes that return the run-time terminal to its default condition any time the CLSDIS function is called. However, when the OPNDIS (open a display file) function is called, it transmits the terminal startup codes for the run-time terminal. What impact, if any, this may have on your application program depends on the design and function of the program itself.

Example:
Line 320 checks to make sure that the file closed without errors.

```
300 DONE: RET.ERR% = CLRSCR          REM Clear terminal screen
310       RET.ERR% = CLSDIS          REM Close the display file
320       CALL DM.ERR(RET.ERR%,CLSDIS$) REM Check for file error
330       STOP                       REM Stop the program
```

CURS Function: Set Cursor Visible/Invisible

Syntax: <string variable> = CURS(<string expression>)

Display Management Function

Explanation:
If the run-time terminal has the ability to turn the cursor ON and OFF, the CURS function can serve to make the cursor visible (ON) or invisible (OFF).

This function works independent of the cursor's location on the screen.

Function Argument:
The function argument (<string expression>) is a single-character indicating the desired setting for the cursor. The value can be zero, one, two, or three, as shown in this table.

Table 7-2. CURS Function Argument Values

Arg. value	Desired result
0	Reset the cursor to the visible state.
1	Set the cursor to the invisible state.
2	Change the current setting; that is to say: visible to invisible, or the reverse.
3	Do not change the current setting.

Return Values:
If the cursor cannot be made visible or invisible, Display Manager ignores the CURS function call. Otherwise, a one position character string is returned indicating the current state of the cursor, as follows:

0 -- The cursor is visible (ON).
1 -- The cursor is invisible (OFF).

Additional Comments:
A display does not have to be on the run-time terminal when the CURS function is called. However, Display Manager must be properly initialized (see the INITDM function description).

Example:
RETDM serves in line 150 to find out what features are available on the terminal. Line 160 checks to see if the cursor can be turned ON and OFF. If it can, it is turned OFF at line 170; otherwise, the terminal is initialized with the cursor ON in line 180 and 190.

```
150    AVAIL.ATTR$ = RETDM            REM Get terminal's attributes
160    IF MID$(AVAIL.ATTR$,1,1) = 1   REM Cursor turn ON/OFF?
170      THEN CALL CURS("1")          REM Yes-turn it off
180      ELSE PRM.ON$ = "0" :         REM No-initialize with
190           PRM.OFF$ = "3"          REM    cursor ON.
```

DISPD Function: Place Display on Screen

Syntax: <integer variable> = DISPD(<integer expression>)

Display Management Function

Explanation:
DISPD retrieves the display that you specify from the open display file, and places it on the run-time terminal screen. To specify which display, you must provide its display reference number.

Cursor Disposition:
The cursor is placed in the bottom-right corner of the screen.

Function Argument:
The function argument (<integer expression>) is the display reference number of the display to be shown. The value must not be less than 1, nor more than 250.

Return Values:
If the display is found in the current display file, its display reference number is returned. If the display cannot be found, a negative value is returned.

Additional Comments:
The DISPD function reads the specified display from the current display file, and places it on the screen of the run-time terminal. All literal fields and visible, initial values for input and output fields are shown. If any video or color attributes were assigned to the fields in the display, they take effect if available on the run-time terminal. Fields assigned the Invisible attribute (in the field's status window) are suppressed.

If the display was designed with the clear screen option, Display Manager automatically clears the screen before the display is shown. Otherwise, the screen is not cleared (unless the CLRSCR function is used), and the display overlays any image currently on the screen. However, your application program can only reference fields in the current display.

Example:
The display file is opened in line 100; line 110 checks to make sure it was opened successfully. A display is called from the display file in line 120 (the display is referenced by the variable ORDER.FORM%); line 130 checks to make sure that the display was found. Line 140 moves the cursor to the first field in the display (using the NXTF function).

```
100    RET.ERR% = OPNDIS("ORDERS.DIS") REM Open the display file
110    CALL DM.ERR(RET.ERR%,OPNDIS$)   REM Check for file error
120    RET.ERR% = DISPD(ORDER.FORM%)   REM Show Order Form Display
130    CALL DM.ERR(RET.ERR%,DISPD$)    REM Check if display found
140    RET.ERR% = NXTF(-10)            REM Move cursor to first field
```

ENDF function: Determine Data Entry Termination Method

Syntax: <integer variable> = ENDF

Field Management Function

Explanation:
The ENDF function returns a value indicating how the end-user terminated data entry for a field. The returned value is always in reference to the most recent use of GETF or UPDF to retrieve information from the field.

Return Values:

An integer value is always returned. The following table lists and explains the possibilities.

Table 7-3. ENDF Return Values

Format: Value
Termination Method Used

0 (ASCII null).
Indicates normal termination. Means that the field was filled while AutoRet was set in its status window, that the RETURN key was pressed, or that an immediate return resulted from retrieving information from an output field.

x
Abnormal termination. The "x" is the ASCII value of the invalid character entered when the BadC or Cursor attribute is set in the field's status window. For example, if the field is defined as numeric only and the letter A is entered, data entry is terminated and the value returned is 65.

-n
-n is a negative number indicating the function key that was pressed when the FKey attribute is set in the field's status window. For example, -1 is returned when function key #1 is pressed.

Additional Comments:
Display Manager resets the ENDF function every time a new display is placed on the terminal screen. This ensures that there is no carry over of conditions from a previous display.

Note that, whatever cursor control keys are used, the up (Ctrl-K) and down (Ctrl-L) keys are mapped into 11 and 10 when the "Cursor" attribute is set in the field's status window. When the Cursor attribute is not set but BadC is, ENDF returns the ASCII value entered by the end-user. These distinctions are important, because knowing the method used to terminate data entry is useful for allowing the end-user to move between fields without ill effects (see the RESF function).

Example:
The GETF function is used at line 250 to retrieve the end-user's input. At line 260, ENDF checks to see if the ESC key was pressed to terminate data entry and signal the end of the program. If it was, lines 270 through 290 close the file, check for errors during the close, and stop the program.

```
250    INPUT$ = GETF                    REM Get field input
260    IF ENDF = 27                     REM ESC key entered?
270      THEN RET.ERR% = CLSDIS :          REM Yes-close display file
280           CALL DM.ERR(RET.ERR%,CLSDIS$)   REM Check for file error
290           STOP                        REM Stop the program
```

GETF Function: Retrieve/Validate Field Input

Syntax: <string variable> = GETF

Field Management Function

Explanation:
GETF retrieves information from the current field, and makes it available to your application program.

Cursor Disposition:
The cursor is left in the column immediately to the right of the field from which the data was retrieved.

Return Values:
GETF can be used to retrieve data from either an input or output field. The function always returns a string variable.

Input Fields

GETF returns the characters entered by the end-user. Only those characters entered are returned. If the field contains an initial value or template characters, these are not returned. (Use the UPDF function to retrieve initial values from a field.)

Each time the end-user types a character into the field, the character is validated according to the validation code set in the field's status window. If data entry is terminated abnormally, all characters entered in the field (except the illegal character) are returned to the application program.

All characters typed into the field are returned until one of the following occurs:

- The RETURN key is pressed.

- Data entry is abnormally terminated (for example, an illegal character is entered).

- The field is filled and the AutoRet code in the field's status window is appropriately set.

If the input field contains initial values, and the end-user does not enter data into the field, a null string is returned.

GETF makes it possible for the end-user to use editing control keys when entering data in an input field. The next table shows these keys, and their resulting action.

Table 7-4. Data entry editing control keys

Format: Key
Result

Right Arrow
Moves cursor right one position in the field. Characters in the field are not changed. If the cursor is in the last position of the field, it does not move.

Left Arrow or Backspace
Moves cursor left one position in the field. Characters in the field are not changed. If the cursor is in the first position of the field, it does not move.

Ctrl-S
Same as left arrow key.

Ctrl-D
Same as right arrow key.

Ctrl-G
Deletes the character under the cursor. All characters to the right of the deleted character (except template characters) shift left one space.

DEL ("<--")
Deletes the character to the left of the cursor. The character under the cursor and those to the right (except template characters) shift left one space. If the cursor is in the first position of the field, the character under the cursor is deleted.

Ctrl-V
Inserts a space at the current cursor location. In the field, the character under the cursor and those to the right (except template characters) shift right one space.

Notes:

1. Refer to the INITDM function description if you want to assign different arrow keys.

2. When using any of the keys in the preceding table that shift characters, any characters shifted beyond the right boundary of a field are lost.

Based on the validation code in the field's status window, GETF returns characters that conform to the specification. For example, if the code specifies integer information, a string of integers is returned. Thus, 123.45 is returned as 123.

Output Fields

When GETF is used in conjunction with an output field, information from the field is returned immediately. That is to say: it is not necessary for the end-user to terminate data entry, since data is not entered into output fields. This is a useful method for reading information from the display.

Example:
Line 130 places the cursor in the next, relative input field. The GETF function in line 150 retrieves information from the field.

```
110 DEF GET.ENTRY
120    STRING GET.ENTRY
130    RET.ERR% = NXTF(2)           REM Curs to next input field
140    CALL DM.ERR(RET.ERR%,NXTF$)    REM Check for errors
150    INP$ = GETF                  REM Get field input
160 CONT: IF ENDF = 27              REM ESC key pressed?
170       THEN RET.ERR% = CLSDIS :   REM Yes-close display file
180            CALL DM.ERR(RET.ERR%,CLSDIS$)  REM Check for file error
190            STOP              REM Stop the program
200     IF ENDF <> 0 AND ENDF <> 26 \ REM No-^char or ^Z pressed?
210       THEN GOTO RETR          REM Yes-go to abnormal end
220     GET.ENTRY = INP$          REM No-get input data
230     RETURN
240 RETR: RET$ = RESF(-1)           REM Save field position
250     CALL DM.ERR(RET.ERR%,PUTF$)   REM Check for errors
260     INP$ = RESF(1)            REM Resume input
270     GOTO CONT                 REM Continue
280 FEND
```

INITDM Function: Initialize Display Manager

Syntax: <integer variable> = INITDM(<string expression>)

Initialization Function

Explanation:
The INITDM function initializes the run-time application program and the run-time terminal for use with Display Manager. This function must be called by your application program prior to calling any other Display Manager function.

Function Argument:
The function argument (<string expression>) contains program attributes and terminal control codes. The string must be in this format, and cannot contain more than 254 characters:

<program attributes>/<terminal control codes>

Although the program attributes portion of the string is optional, the slash (/) and the terminal control codes are always required. Note that the terminal control codes in the TERMS.DM file already have the slash as a prefix.

Program Attributes

Your application program can pass a string of up to 5 program attributes to Display Manager at run-time. These attributes each have default values that are used unless you change them by passing new ones. The following table shows the program attribute that corresponds to each position in the string.

Table 7-5. Program attributes string

Position	Attribute
1	Money Symbol
2	Decimal Delimiter
3	Alphabetic Character Set
4	Cursor Movement Keys
5	Clock Set

The following describes the optional settings for these attributes. The default value for all attributes is A. Thus, the default string is AAAAA. This sets the money symbol to a dollar sign ($), the decimal delimiter to period (.), the alphabetic character set to the ASCII standard, cursor movement keys to both sets, and the clock to 4-MHz (megahertz).

1) Money Symbol

The character specified in this attribute is displayed in an input or output field that has been defined as a money field. The default character is a dollar sign ($), but any printable character can be used.

A
Means the dollar sign ($) should be used.

(x)
If you want money fields to be displayed with something other than the dollar sign, pass the character that you want to use as the attribute value.

2) Decimal Delimiter

The decimal delimiter attribute tells Display Manager how to show fractional numbers in a decimal format. This attribute takes effect when a field is displayed using the PUTF (Put Field) function.

A
Tells Display Manager to use the American standard, where a period is the decimal delimiter, and a comma is the separator. This causes numbers to print as follows:

123,456,789.01

E
Tells Display Manager to use the European standard, where a comma is the decimal delimiter, and a period is the separator. Numbers print as follows:

123.456.789,01

3) Alphabetic Character Set

The standard set of alphabetic characters includes ASCII characters a-z and A-Z. Display Manager considers only those characters in this set as acceptable alphabetic characters when performing validation of information entered by the end-user (see the "Input Field Validation Codes" table in Section 6). This attribute makes it possible to extend the alphabetic character set up to 3 characters following z and Z. The characters that become available are dependent on the terminal in use. For example, European terminals often have accented characters in these positions.

A
Means only the characters in the standard ASCII set are acceptable.

B
Extends the set by 1 character.

C
Extends the set by 2 characters.

D
Extends the set by 3 characters.

4) Cursor Movement Keys

You can use 2 different sets of cursor movement keys with Display Manager: the ANSI-standard keys, and the Display Manager keys.

A
Means both sets may be used.

B
Means only the ANSI-standard keys may be used (Ctrl-K, Ctrl-H, Ctrl-L, and Ctrl-J).

C
Means only the Display Manager set may be used (Ctrl-E, Ctrl-S, Ctrl-D, and Ctrl-X).

5) Clock Set

This attribute selects the clock speed of the run-time computer. The default value (4-MHz) is normally adequate for most computers. You may wish to adjust this rate if there is a timing problem with the run-time terminal, such as clearing the screen or positioning the cursor. If the run-time computer is a very slow system based on Z-80 microchips, setting the clock to a lower rate may clear the screen and position the cursor on some terminals a little faster.

A 4-MHz
B 1-MHz
C 2-MHz

D 3-MHz
E 5-MHz
etc.

Terminal Control Codes

The terminal control codes portion of the argument string contains the codes that Display Manager needs to initialize the run-time terminal. The codes tell Display Manager what features are available on the terminal, and how to activate/deactive them.

Please read Section 3 and Appendix C for additional information on setting up control codes for various run-time terminals.

Return Values:
Zero is returned if the application program and the run-time terminal are successfully initialized.

Display Manager checks the control codes for the run-time terminal. If there are errors, a short message is displayed on the run-time terminal, and one of the negative numbers listed and explained in the next table is returned to your application program. The error descriptions in the table make reference to the terminal control code example string shown here:

```
 Number of non-null groups
 |
/2 ABBECDAABABABART GGGHAAAAKJAIFHPPZ6
  +--------------+ +----------------+
     Group 1          Group 30
```

Figure 7-2.Terminal Control Code Example String

The "2" following the slash (/) indicates that there are 2 groups in the string containing non-null characters, Groups 1 and 30. After the first group, the remaining groups can be in any order, because the group number is actually part of the code.

Table 7-6. INITDM Run-time Errors

Format: Returned Value
Description of Error

-1 to -5
One of the code groups in the string contains an unacceptable code. The value returned indicates which group: for example, -2 indicates the second group, group 30 in the example string. See Appendix A for a discussion of what constitutes acceptable codes.

-40
Indicates that the number following the slash is invalid or missing.

-50

The code string contains non-blank characters beyond the end. In the preceding example, 6 marks the end of the code string.

-100
Indicates that the slash, which must precede each code string, is missing.

Additional Comments:
The INITDM function can only be called once by your application program.

Example:
In this example, CURRENT.TRM is assumed to contain the terminal control codes for the terminal being used. Initialization occurs in line 50. If initialization is not successful (line 60), an error message is displayed (line 70) and the program is stopped.

```
010    IF END #1 THEN ERR1              REM If no terminal file, error
020    OPEN "CURRENT.TRM" AS 1          REM Open terminal ctl code file
030    READ #1;TERM$                    REM Read terminal ctl code string
040    CLOSE 1                          REM Close the file
050    RET.ERR% = INITDM(TERM$)         REM Initialize library a terminal
060    IF RET.ERR% < 0                  REM Check for initialize error
070      THEN PRINT "Bad DM code" :     REM Yes-print error message
080           STOP                      REM Stop the program
```

NXTF Function: Place Cursor in Relative Field

Syntax: <integer variable> = NXTF(<integer expression>)

Field Management Function

Explanation:
NXTF places the cursor in a specified field of the display. The field you want to receive the cursor is referenced relative to the current field.

Following the NXTF function, the field receiving the cursor becomes the current field.

Cursor Disposition:
The cursor is located in the first position of the receiving field. However, if the specified field cannot be found, the cursor is not moved.

Function Argument:
The function argument (<integer expression>) specifies the field to which the cursor is to be moved, and its type. The following table shows the acceptable argument values, and their results. Remember that the receiving field is always referenced relative to the current field.

Table 7-7. NXTF argument values

Value	Cursor Moves To...
1	NEXT input or output field
2	NEXT input field

```
  3   NEXT output field
 10   LAST input or output field in display
 20   LAST input field in display
 30   LAST output field in display
 -1   PREVIOUS input or output field
 -2   PREVIOUS input field
 -3   PREVIOUS output field
-10   FIRST input or output field
-20   FIRST input field
-30   FIRST output field
```

Return Values:
If the specified field is located, the field reference number of the new current field is returned; otherwise, a negative value is returned.

Additional Comments:
The cursor moves according to the order of fields on the screen, and without regard to field reference numbers. Consequently, it is not necessary for your application program to use specific numbers to reference fields (although you can use the POSF function to place the cursor according to field reference numbers).

Example:
This example places the cursor in the next input field relative to the current field.

```
100    RET.ERR% = NXTF(2)            REM Cursor to next input field
110    CALL DM.ERR(RET.ERR%,NXTF$)    REM Check for errors
120    INP$ = GETF                   REM Get field input
```

The next example places the cursor in the previous output field relative to the current field. A total is computed, and then displayed in the output field.

```
200    RET.ERR% = NXTF(3)            REM Cursor to prev output field
210    CALL DM.ERR(RET.ERR%,NXTF$)    REM Check for errors
220    TOTAL(ORDER.NO%) = STR$(VAL(QTY(ORDER.NO%))  REM Compute the total
230                 * VAL(PRICE.EA(ORDER.NO%)))
240    RET.ERR% = PUTF(TOTAL(ORDER.NO%))  REM Display data in field
250    CALL DM.ERR(RET.ERR%,PUTF$)    REM Check for error
```

The final example places the cursor in the first field of the display.

```
300    RET.ERR% = NXTF(-10)          REM Cursor to first field
310    CALL DM.ERR(RET.ERR%,NXTF$)    REM Check for errors
320    INP$ = GETF                   REM Get field input
```

OPNDIS Function: Open Display File

Syntax: <integer variable> = OPNDIS(<string expression>)

Display Management Function

Explanation:
Opens a display file. Only one display file can be open at any given time. So, if a display file is already open when this function is called, the open file is automatically closed before this function is performed.

If there is a display currently on the run-time terminal, it is not affected by this function, unless the new display has the Clear Screen global value set. Note, however, that your application program cannot access any of the fields in the original display.

Function Argument:
The function argument (<string expression>) is the name of the display file to be opened. If the file is on a disk in a drive other than your currently-logged drive, precede the name with the appropriate drive specifier. For example, B:PAYROLL.DIS opens the file on drive B.

Return Values:
If the display file is opened successfully, a zero is returned; otherwise, a negative value is returned.

Additional Comments:
A display file must be opened with the OPNDIS function before your application program can use any of the displays contained therein. But, once a display file is open, you can use any and all displays from that file any number of times, without reopening it.

Your program must call the INITDM function before using OPNDIS.

Any time the OPNDIS function is called, Display Manager sends the startup codes to the run-time terminal.

Example:
This example opens a display file named "ORDERS.DIS".

```
150    RET.ERR% = OPNDIS("ORDERS.DIS") REM Open ORDERS.DIS file
160    CALL DM.ERR(RET.ERR%,OPNDIS$)   REM Check for file error
```

POSF Function: Place Cursor in Specific Field

Syntax: <integer variable> = POSF(<integer expression>)

Field Management Function

Explanation:
POSF places the cursor in a specified field of the display. The receiving field is specified by its field reference number, and becomes the current field in the display.

Cursor Disposition:
The cursor is located in the first position of the specified field. However, if the specified field cannot be found, the cursor's location is not changed.

Function Argument:

The function argument (<integer expression>) is the field reference number of the field that is to receive the cursor. Field numbers can range from 1 to 250.

Return Values:
If the specified field is located or zero is passed as the function argument, the field reference number of the current field is returned. If zero is passed as the function argument and no field is current (that is to say: the cursor is not in a field), zero is returned.

Additional Comments:
If the specified field cannot be located, a run-time error results (see Section 8).

See the NXTF function for placing the cursor in a field relative to the current field.

Example:
In this example, line 120 stores the current field number. Line 130 moves the cursor to the field specified by FIELD.NO%. Line 160 returns the cursor to the original field.

```
120    CURR.FLD% = POSF(0)           REM Save current field number
130    RET.ERR% = POSF(FIELD.NO%)    REM Move cursor to specific field
140    ...                   REM   (Other processing)
150    ...                   REM
160    RET.ERR% = POSF(CURR.FLD%)    REM Cursor back to original field
```

In the next example, the cursor is moved from field to field in the display according to field reference numbers. This routine assumes the fields are numbered sequentially in increments of 1.

```
240    MAX.FLDS% = NXTF(10)    REM Save number of last field
250    CNT% = 0                REM Initialize loop index
260 NEXT: RET.ERR% = POSF(CNT% + 1) REM Move cursor to next field
270    CALL DM.ERR(RET.ERR%,POSF$) REM Check for errors
280    ...                REM   (Other processing)
290    ...                REM
350    CNT% = CNT% + 1        REM Add one to index
310    IF CNT% < MAX.FLDS%    REM Last field reached?
320      THEN GOTO NEXT       REM No-Continue loop
```

PUTF Function: Display Data in Field

Syntax: <integer variable> = PUTF(<string expression>)

Field Management Function

Explanation:
PUTF displays data in the current field according to its specified format. The output format of a field is specified in its status window while using the Editor to create the display.

The PUTF function displays string characters in a display field, even when the Format code in the field's status window is N (specifying Numeric data).

Cursor Disposition:
The cursor is placed in the column immediately to the right of the last character in the field. If the specified field cannot be found, the cursor is not moved.

Function Argument:
The function argument (<string expression>) is the information to be displayed in the field. The string must not contain more than 132 characters.

Return Values:
PUTF returns zero if the information is successfully displayed in the field; otherwise, a negative value is returned. A negative value might result if, for example, the information being placed in a number-type field is too large for that field (in which case, the field is filled with asterisks).

Additional Comments:
The PUTF function erases any data that may have been left in the field from a previous operation.

Example:
Line 100 moves the cursor to the next output field in the display (relative to the current field). Line 120 computes a total to show in the field. Line 130 displays the total in the field.

```
100    RET.ERR% = NXTF(3)            REM Cursor to next output field
110    CALL DM.ERR(RET.ERR%,NXTF$)    REM Check for error/field found?
120    TOTAL = STR$(VAL(QTY) * VAL(PRICE.EA))  REM Compute total
130    RET.ERR% = PUTF(TOTAL)         REM Display total in field
140    CALL DM.ERR(RET.ERR%,PUTF$)    REM Check for error/good data?
```

RESF Function: Resume Data Entry

Syntax: <string variable> = RESF(<integer expression>)

Field Management Function

Explanation:
The RESF function provides a way for your program to interrupt data entry, save the information entered into the field thus far, and then resume data entry exactly where it was interrupted.

For example, assume that the end-user is entering information into a field whose validation code (in the status window) specifies that only numeric data can be entered. This is a five-position field and the end-user enters the following information: "012F". When the F is entered, abnormal data entry termination is forced. Your program can call the ENDF function to determine why data entry was terminated. The RESF function can then be called to save the valid information entered into the field (12) without saving the invalid character (F). An error message can then be displayed, explaining why data entry was terminated (based on the value returned by ENDF). Finally, RESF can

be called again, to return the cursor to the original input field and resume data entry at the position where the invalid character was entered.

RESF resumes data entry using the GETF or UPDF function, whichever was in use at the time of abnormal termination.

Cursor Disposition:
The cursor's location following this function is dependent on the value passed in the function argument (see the next paragraph).

Function Argument:
The function argument (<integer expression>) is a single-character integer specifying one of 8 storage locations. If the argument contains a negative value (-1 to -8), Display Manager saves the information entered up to the point of termination, and returns an empty string. It also remembers the field in which data was being entered. The RESF function can subsequently be called with a corresponding positive value (1 to 8) to cause data entry to resume exactly where it was terminated.

For example, if your program detects abnormal termination, but you want the end-user to be able to return to the field and resume entering data, you would first call RESF with a negative value. When you want to return the user to the original field for resumption, call RESF with a corresponding positive value.

Return Values:
RESF returns the same value as that returned by the GETF or PUTF function originally used to retrieve data from the field.

Additional Comments:
RESF with a positive argument continues data entry for a particular field using the rules associated with either the GETF or UPDF function, depending on which was used to retrieve the data. The cursor is positioned at the location where data entry terminated to continue the input. The value returned by RESP is the character string entered by the end-user.

Example:
Line 100 retrieves the input entered by the end-user. The following 3 lines, 110 to 130, would check for abnormal data entry termination using the ENDF function. If abnormal termination occurs because of an illegal character, as determined by the ENDF function, data entry resumes by saving the current field information (line 140), and resuming input as shown in line 150.

```
100   INP$ = GETF     REM Get input from field
110   ...
120   ...           REM  (Other processing)
130   ...
140   RET$ = RESF(-1) REM Save field no. and valid data
150   INP$ = RESF(1)  REM write old data in field/resume input
```

RETDM Function: Return Run-time Terminal Attributes

Syntax: <string variable> = RETDM

Explanation:
Returns a string of 16 separate values. However, only the first 11 values are currently in use, and they indicate the following:

- the features available on the run-time terminal
- the version of the Display Manager Run-time Library currently in use

Return Values:
Each position in the string variable returned by the RETDM function pertains to a standard terminal feature supported by Display Manager. The following table explains the individual string positions and their corresponding features.

Table 7-8. RETDM terminal features

Pos.	Feature
1	Cursor Visibility
2	Half Intensity
3	Inverse Video
4	Flashing
5	Underline
6	User-defined #1
7	User-defined #2
8	User-defined #3
9	Function Keys
10	Run-time Version Number
11	Color/Monochrome Indicator

The values returned in positions 1 through 9 indicate whether or not the corresponding feature is available on the run-time terminal. It further indicates whether or not the feature requires a reserved space on either side of a field (meaning that the terminal uses the Plant method for activating video attributes). Each position can contain an ASCII value of 0, 1, or 2:

0: The feature is not available on the run-time terminal.

1: The feature is available, and does not require a space on either side of the field.

2: The feature is available, and does require a space on either side of the field.

Your application program can examine the returned string to determine whether or not to activate a particular feature. For example, you might want to activate inverse video if that feature is available, or underlining if it is not. In such a case, activate the feature only if the corresponding position contains a 1; do not activate it if the position contains a 0 or 2.

The 9th position indicates the number of function keys available. ASCII zero ("0") indicates none are available, "1" indicates that one function key is available, and so forth.

The 10th position indicates the version and release number of the Run-time Library in use. A value ranging from 0 to 127 (7F hexadecimal) might be returned. The leftmost digit is the version number; the rightmost is the release number. For example, hexadecimal 11 signifies version 1, release 1. This value is significant, since displays created with one version of Display Manager generally do not work with a different version of the Run-time Library.

The last position indicates whether the run-time terminal is monochrome or color equipped. If monochrome, ASCII 0 is returned; if color, ASCII 1 is returned.

For example, if the RETDM function returns the string

10110000000

it indicates that the invisible cursor, inverse video, and flashing cursor features are available. Half intensity, underlining, and user-defined features are not available, no function keys are available, and version zero/release zero of the run-time library is being used. Finally, it indicates that the run-time terminal is a monochrome-type.

Example:
In this example, the features available on the run-time terminal are returned to the application program in line 150. Line 160 checks the returned string, to see if the inverse video feature is available. If it is, the feature is used to highlight prompts (lines 170 and 180).

```
150    AVAIL.ATTR$ = RETDM            REM Get terminal attributes
160    IF MID$(AVAIL.ATTR$,3,1) <> "0" REM Is inverse video supported?
170      THEN PRM.ON$ = "031" :      REM Yes-use the feature to
180           PRM.OFF$ = "330"       REM     highlight prompts.
```

RETF Function: Return Field Position, Length, and Type

Syntax: <string variable> - RETF

Field Management Function

Explanation:
This function returns a value indicating the position, length, and type (input or output) of the current field.

Return Values:
RETF returns a 16-character string. However, only the first 8 characters are currently used, and they contain information about the current field. The next table explains the value of each character position.

Table 7-9. Field information from RETF

Format: Position
Meaning

1
An ASCII value ranging from 1 to 255, indicating the ROW number in which the field is located.

2
(Not used; contains ASCII 0.)

3
An ASCII value ranging from 1 to 255, indicating the COLUMN number where the field begins.

4
(Not used; contains ASCII 0.)

5
An ASCII value ranging from 1 to 255, indicating the LENGTH (number of columns) of the field.

6
(Not used; contains ASCII 0.)

7
Position seven indicates whether or not a space exists on either side of the field: 0 indicates no space is on either side of the field. 1 indicates a space is on either side (meaning a Plant-type attribute can be set for the field).

8
Indicates whether the field was defined as an input or output type: "I" indicates an input field, "O" (letter O, not zero) indicates an output field.

Example:
The RETF function in this example determines the length of a particular field. Line 360 obtains the field information. Line 370 computes the length of the field.

```
300 DEF WRITEF(OUT$)
310    RET.ERR% = NXTF(2)            REM Cursor to next input field
320    CALL DM.ERR(RET.ERR%,NXTF$)    REM Check for error
330    ATTR$ = SETF("0")             REM Make field visible
340    RET.ERR% = PUTF(OUT$)         REM Display old data in field
350    CALL DM.ERR(RET.ERR%,PUTF$)    REM Check for error
360    RET$ = RETF                   REM Get field values
370    FIELD.SZ% = ASC(MID$(RET$,5,1)) REM Determine field size
390    IF FIELD.SZ% = 1 \            REM Check field size
400       THEN RET.ERR% = NXTF(3) : \  REM Curs to next output field
410       CALL DM.ERR(RET.ERR%,NXTF$) : \  REM Check for error
420       RET.ERR% = PUTF(MID$(OUT$,2,LEN(OUT$))) : \   REM Show data
430       CALL DM.ERR(RET.ERR%,PUTF$)  REM Check for error
440 FEND
```

SETF Function: Set Field Attributes

Syntax: <string expression> = SETF(<string expression>)

Field Management Function

Explanation:
This function can be used to set the video or color attributes of a field during run-time. These attributes are originally set in the field's status window during the display design with the Editor. This function makes it possible to override the original attributes.

Function Argument:
The function argument (<string expression>) is a 16-character string. Each character in the string refers to a specific attribute on the run-time terminal that can be set for the field. The next table shows the meaning of each character in the string, along with its normal and special setting.

Table 7-10. SETF argument values

Pos.	Attribute	Nrm	Spc
VIDEO			
1	Visibility	Y	N
2	Half Intensity	N	Y
3	Reverse Video	N	Y
4	Flashing Field	N	Y
5	Underlining	N	Y
6	User-defined #1	N	Y
7	User-defined #2	N	Y
8	User-defined #3	N	Y
COLOR			
9	Flashing	N	Y
Background Color Codes			
10	RED	N	Y
11	GREEN	N	Y
12	BLUE	N	Y
Foreground Color Codes			
13	Intensity	N	Y
14	Red	Y	N
15	Green	Y	N
16	Blue	Y	N

The value of each position in the string indicates how you want the attribute to be set, as follows:

0 Normal setting.
1 Special setting.
2 Change current setting (normal to special, or the reverse).
3 Do not change current setting.

The argument is evaluated from left to right. If it contains less than 16

characters, only the attributes for which a new value is sent are changed; the remaining attributes are not affected. Characters beyond 16 are ignored.

Return Values:
SETF returns a 16-character string indicating how attributes are currently set on the run-time terminal for this field. The characters in this string correspond to the attributes shown in the preceding table. If the corresponding position in the string contains 0, the attribute is in the normal state; if it contains 1, it is in the special state.

Additional Comments:
Use the POSF or NXTF function to locate a particular field in a display. Use the POSF function with a zero function argument to determine the number of the current field.

Example:
The following example is a portion of a routine that accepts input from the display on the run-time terminal. The attributes available on the terminal were determined previously using the RETDM function. The attributes for any given field can be changed using the SETF function, as shown in lines 120 and 210.

```
100    RET.ERR% = NXTF(2)            REM To next input field
110    CALL DM.ERR(RET.ERR%,NXTF$)    REM Check for error
120    ATTR$ = SETF("0")             REM Turn prompt ON
130    INP$ = GETF                   REM Get field input
140 CONT: IF ENDF = 27 \             REM ESC key to exit?
150        THEN RET.ERR% = CLSDIS :   REM Yes-close display file
160             CALL DM.ERR(RET.ERR%,CLSDIS$) :  REM Check for file errors
170             STOP                 REM Stop the program
180     IF ENDF <> 0 AND ENDF <> 26 \ REM Ctrl char (not ^Z)?
190        THEN GOTO RETR            REM Yes-ignore char, continue
200     GET.ENTRY = INP$             REM No-get input
210     ATTR$ = SETF("1")            REM Turn prompt OFF
220     RETURN
230 RETR: RET$ = RESF(-1)            REM Save field no. and data
260     INP$ = RESF(1)               REM Write old data in field
270     GOTO CONT                    REM Continue
```

UPDF Function: Retrieve/Validate Field Input

Syntax: <string variable> = UPDF

Field Management Function

Explanation:
UPDF retrieves information from the current field, and makes it available to your application program.

UPDF is similar to the GETF function except that, if the field contains an initial value, the value is returned with UPDF, whereas it is not returned with GETF.

Cursor Disposition:
The cursor is left in the column to the right of the field from which the data was retrieved.

Return Values:
UPDF can be used to retrieve data from either an input or output field.

Input Fields

UPDF returns the characters entered by the end-user. If the field contains an initial value, the value is returned even if the end-user does not enter any characters into the field. However, if the field contains template characters, these are not returned.

Each time the end-user types a character into the field, it is validated according to the Validate code set in the field's status window (see Table 4-3, "Input Field Validation Codes"). If data entry is terminated abnormally, all characters entered in the field (excluding the illegal character) are returned to the application program.

UPDF returns all characters entered into the input field until the occurrence of one of the following:

- The RETURN key is pressed.

- Data entry is abnormally terminated (for example, an illegal character is entered).

- The field is filled and the AutoRet code in the field in status window is appropriately set.

UPDF makes it possible for the end-user to use editing control keys when entering data in an input field. The table in the GETF function description shows these keys and their interpretation.

Output Fields

When UPDF is used in conjunction with an output field, characters in the field are returned immediately. That is to say: it is not necessary for the end-user to terminate data entry, since data is not entered into output fields.

Additional Comments:
UPDF is used to access updated information in a field. It is your responsibility to ensure that the initial value in a field is the type that your application program expects. The UPDF function does not validate the entire field; it checks only those characters entered by the end-user. An exception to this is if the validation code specifies decimal data. UPDF allows a decimal point to be entered if it is to the left of an existing decimal point in the field. The function removes the existing decimal point

without affecting other characters in the field. If the end-user attempts to enter a decimal point to the right of an existing one, Display Manager does not accept it. However, this does not cause abnormal termination of data entry.

Example:
This example shows UPDF used to accept data input to a field in line 250. Prior to calling UPDF, an initial value was placed in the field by a PUTF statement (line 220).

```
200    RET.ERR% = NXTF(2)            REM Cursor to next input field
210    CALL DM.ERR(RET.ERR%,NXTF$)    REM Check for errors
220    RET.ERR% = PUTF(PRICE$(PART%))  REM Display initial value
230    CALL DM.ERR(RET.ERR%,PUTF$)    REM Check for errors
240    ATTR$ = SETF(PRM.ON$)          REM Turn prompt ON
250 PTRY:PRICE.EA(ORDER.NO%) = UPDF    REM Get input data
255 REM ------------------------------------------------------
260 REM This field does not trap bad characters; ESC not trapped.
265 REM ------------------------------------------------------
270    ATTR$ = SETF(PRM.OFF$)         REM Turn prompt OFF
```

Section 8: Run-time Environment

The run-time environment represents the culmination of work accomplished in the terminal setup, Editor, and applications programming environments. The following figure "Run-time Environment" illustrates the normal run-time environment, though many variations are possible.

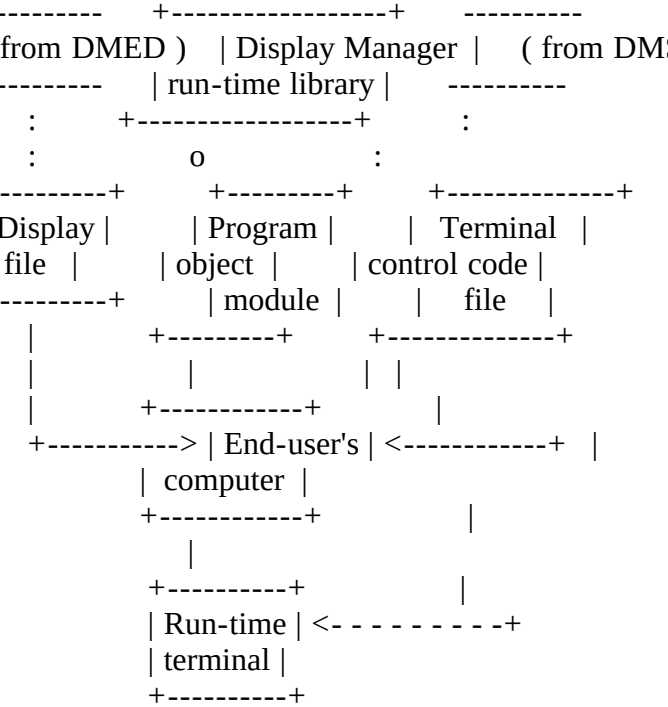


Figure 8-1. Run-time Environment

The Terminal Control Code File created in the terminal setup environment

contains the codes that your application program needs to work with the run-time terminals. At run-time, your program can determine the run-time terminal in use, read the terminal control codes for that terminal from this file, then pass the codes via the INITDM function to initialize the terminal.

There is a variation to this method of initializing the run-time terminal. Your Display Manager distribution disks contain a source program named INSTALL.BAS. This program is written in CBASIC Compiler, and is designed to help the end-user initialize the run-time terminal for use with your application program. INSTALL.BAS reads the terminal control codes from the TERMS.DM file (rather than from the Terminal Control Code File that you create with DMSET), and uses those codes to initialize the terminal. If you are developing an application for use on a variety of terminals, this program may be of use to you.

You can change the INSTALL.BAS program any way you wish, or use it as distributed. Simply compile and link the program, and provide whatever documentation you feel that the end-user may require. Distribute the object module, documentation, and a copy of TERMS.DM with your application.

The program object module is created in the application programming environment, and linked to the Display Manager Run-time Library to include the routines required by your application. Of course, your application can consist of one or more program object modules.

The final component in this environment is the display file containing all of the displays to be used in your application. This file is created in the Editor environment. There can be as many display files as disk space permits on the run-time computer.

All of these components (display files, program object modules, terminal control code file, and the optional INSTALL.BAS program and TERMS.DM file) can be distributed for use on the end-user's run-time computer.

Note: You cannot distribute the Editor program.

Errors can occur at run-time if your application program does not use the routines in the Display Manager Run-time Library correctly. However, if you thoroughly test your applications before they are distributed, and your program adequately checks the values returned by the routines, the end-user should never encounter such errors.

When these errors do occur, they are considered fatal to the application program. So, Display Manager automatically closes any files that are open, displays a message indicating the nature and origin of the error, then returns control to your operating system.

The error message displayed in these cases contains a code indicating the function (or routine) that was called when the error occurred, as well as an indication of what caused the error. Here is an example of a run-time error message:

```
Display Manager error: b3  CLSDIS    no display
```

Note the code portion of the message: "b3". The lower case letter indicates the function that caused the error. Each Display Manager function is assigned a separate letter, as shown in the next table.

Table 8-1. Run-time error function codes

Letter	Function	Letter	Function
a	CLRSCR	i	OPNDIS
b	CLSDIS	j	POSP
c	CURS	k	PUTF
d	DISPD	l	RESP
e	ENDF	m	RETDM
f	GETF	n	RETF
g	INITDM	o	SETF
h	NXTF	p	UPDF

The letter code precedes a hexadecimal value (in the above example: 3) indicating the nature of the error. The following table lists the possible values and their meanings.

Table 8-2. Run-time error values

Format: Value
Meaning

1
Your program called a Display Manager function prior to calling the INITDM function. INITDM must be the first Display Manager function called in your application program.

2
No Display Manager display file is currently open. You must open a display file (using the OPNDIS function) before calling any display management functions.

3
There is no current display. This might occur if you attempt to use a field management function in your program before calling the DISPD function.

4
No current field is in the display. This usually occurs if you attempt to use a field management function before calling the NXTF function.

5
You made a second attempt to use the INITDM function. You can use INITDM only once in any given program.

6
The value of the argument passed in a RESF function is less than 1 or more than 8. Only values within this range are permitted.

7
You called the RESF function with a positive argument value before calling it

with a negative argument value. You must call RESF with a negative argument value to save an input field before resuming data entry with a positive argument value.

8
Insufficient memory is available on your computer for showing the displays as you have designed them. You might need to use overlays or fewer variables in your program.

9
You tried to use the POSF function to place the cursor in a non-existent field. The error message shows the field reference number that you attempted to use. Consider using the NXTF function, instead of POSF.

10
You tried to use the wrong version of a display file. This occurs when you create a display file with one version of Display Manager, and use it with a different one.

11
You tried to use a file that is not a valid display file. Check to ensure that you have used the correct filename.

Some additional run-time errors occur only with the INITDM function. Section 7 lists and explains these errors under the description of that function.

Appendix A: Terminal Control Codes

This appendix contains information about the terminal control codes used by Display Manager.

Although most terminals differ greatly in their control characteristics and features, Display Manager can operate with any terminal, as long as its control codes are known, and it meets the minimum requirements listed in Appendix B.

A.1 TERMS.DM FILE

Your Display Manager distribution disks contain a file named TERMS.DM. As distributed, this file contains the terminal control codes for most of the more common terminals on today's market (1984). Display Manager can only be made to work with a terminal if its control codes are contained in TERMS.DM. If you attempt to run Display Manager with a terminal not in TERMS.DM, or with the incorrect control codes, your results are unpredictable.

TERMS.DM can contain 2 different categories of terminals, Display Manager-supported, and user-supported.

A.1.1 Display Manager-supported Terminals

These are terminals originally contained in TERMS.DM as distributed by Digital Research. Display Manager-supported terminals are protected in TERMS.DM. You cannot add terminals in this category, change their control codes, or delete them from the file.

The table "Display Manager-supported Terminals", which appears later in this section, lists the Display Manager-supported terminals contained in TERMS.DM. Your TERMS.DM file may vary slightly from the table. Use the DMSET program to examine the file and determine precisely what terminal control codes are available (see Section 3 and Appendix C).

A.1.2 User-supported Terminals

You can add control codes to TERMS.DM. Any codes you add fall into the user-supported terminal category. You can also make changes to terminals in this category, and delete them from the file if necessary. This category is provided so that you can adapt Display Manager to work with a terminal not originally in TERMS.DM. Every terminal used with Display Manager, however, must meet the minimum requirements of an addressable cursor and a clear screen command.

User-supported terminals can be added to, changed, or deleted from TERMS.DM as described in Appendix C of this manual.

An empty table at the end of this section is for you to keep a record of the user-supported terminals that you set up in TERMS.DM. Run the DMSET program to examine the control codes for these terminals.

Every Display Manager- and user-supported terminal in TERMS.DM has associated with it the terminal control codes that Display Manager needs to make the terminal function as expected (either at run-time or while using the Editor). The terminal control code string has a specific structure which must be observed.

A.2 TERMINAL CONTROL CODE STRUCTURES

Terminal control codes appear as a string of characters with delimiters to separate the string into groups as required. The number of codes and code groups in the string (and, consequently, the string's length) can differ for each terminal. However, the length of the string must not exceed 254 characters.

Here is an example of a terminal control code string. It is shown in ASCII format for readability.

```
 +-- indicates the number of code groups
 |
/3 FHPNADIJDFAMABCPFDZ3 GCJKIDPCBKX7 MAPEAEBEPNR9
```

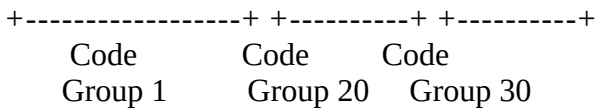


Figure A-1. Example of Terminal Control Code

Each code group in the string contains letter codes ranging from A to P. At the end of each group are error-detecting characters ranging from R to Z and 3 to 9. Display Manager uses the error-detecting characters at run-time to determine if it has the correct codes for the terminal in use. Your applications program must pass the correct terminal control code string to Display Manager at run-time using the INITDM function (described in Section 7).

Three code groups are present in the preceding example:

1) Code Group 1 contains the codes for clearing the screen, positioning the cursor, determining the screen size, and starting up the terminal. This code group is required in all terminal control code strings.

2) Code Group 20 contains the codes for activating standard and user-defined attributes, such as inverse video, half intensity, and graphics. This code group is optional.

3) Code Group 30 contains the codes for activating any function keys on the terminal. This code group is also optional.

Every terminal control code string must have a slash (/) as the first character. A number follows the slash, indicating how many code groups the string contains. Code group one must always be the first group in the string; other groups can follow in any order.

Blanks can appear anywhere in the string, to make it easier to read. If you need to create your own terminal control code strings, do not insert newline (ASCII null = 00H) characters in the string, as they might be treated as the end of the code.

When you set up a user-supported terminal in TERMS.DM, you can prefix the control codes with the terminal name. At run-time, your application program can display the terminal name for the end-user, and request verification that the proper run-time terminal is being used. The name can contain up to 21 characters.

A.3 DISPLAY MANAGER-SUPPORTED TERMINALS

When you use a terminal listed in the following table, you must place the terminal in conversational (or character) mode, and turn OFF all editing features, such as insert and delete. Note also that, if the terminal screen has protected areas, they can affect the appearance of the display. Be careful to avoid switching the terminal OFF during run-time, as this might reset special features that are incompatible with Display Manager.

If one of the codes in TERMS.DM does not work correctly for your terminal, check to make certain that you selected the correct model, and that all the special features have been turned OFF. If it still does not work, find out if you have any customizations that affect the terminal's operation, such as special DIP switches or add-in boards. Turn these switches and features OFF, or run the DMSET program to change the terminal control codes, so that they handle the extra features properly (see Section 3 and Appendix C).

Table A-1. Display Manager-supported terminals

Manufacturer	Model Number/Name
ABM	80
ADDS	40 VT
	Regent 20
	Regent 25
	Regent 40
	Viewpoints 60
	Viewpoints A-2
Apple	Smarterm
	Video Term
Beehive Int.	DM+
	DM Basic
	DM Standard
	DM 5
	DM 5A
	DM 5B
	DM 10
	DM 20
	DM 30
	DM 1A
	Micro Bee 1
	Micro Bee 2
Control Data	721-20
	721-30 (Cyber)
	System 110
cromemco	3102
DEC	VT 52
	VT 100
	VT 100AB (80 Column)
	VT 100AB (80 Column:CP/M-86)
	Rainbow (80 Column)
	Rainbow (80 Column:CP/M-86)
Direct	800
Fujitsu	Micro 16 (CP/M-86)
Hazeltine	1400
	1410
	1420
	1421
	1500
	1510
	1520
	1552
	Esprit

Executive 80 (80 column) (models 20 and 30)
Executive 80 (132 column) (models 20 and 30)
Heath H-19
H-19 (CP/M-86)
H-89
Hewlett-Packard 2382A
2621B
2624B
2626B
Business 125
IBM PC Monochrome (CP/M-86)
PC Color (CP/M-86)
ISC 8001
Lear-Siegler ADM-3A
ADM-5
ADM-31
ADM-42
Micro-Term ACT IV
ACT V
ACT V-A
Osborne System 1
Radio Shack Model II (Pickles and Trout)
Soroc IQ 120
IQ 130
IQ 140
IQ 150
Teleray Model 10 DG
Model 10 MP
Model 10 V52
Model 12
Model 100 (80 column)
Model 100 (132 column)
TeleVideo 910+
912
920
950
Toshiba T100
PASOPIA
Vector Graphics 1600
2600
2800
3005
3032
5005
MZ
System B
Vector 3
Vector 4
Vector 5
Visual Technology
Visual 100
Visual 200
Visual 300
Visual 400

Xerox	820
	860
Zenith	Z-19
	Z-19 (CP/M-86)
	Z-89

A.4 USER-SUPPORTED TERMINALS

This section is provided to help you keep a physical record of user-supported terminals that you add to TERMS.DM. You can photocopy these pages if you need additional space.

Table A-2. User-supported terminals

Manufacturer	Model Number/Name

Appendix B: Summary of Restrictions and Limitations

This appendix summarizes the restrictions and limitations that you must observe when setting up terminals for use with Display Manager, using the Editor, and coding your application programs.

B.1 TERMINALS

Minimum requirements:
Able to clear the screen on command.
Able to position the cursor directly.

Maximum number of rows on CRT:
48 (ROCHE> Not a full page: 66 rows... Too bad!)

Maximum number of columns in a row:
132

Maximum number of positions on CRT screen:
3,840 (if run-time computer has at least 64K RAM).
1,920 (if run-time computer has minimum 48K RAM).

Minimum number of positions on CRT screen:
24 rows by 52 columns.

Maximum number of video attributes that Display Manager can use on a given terminal:
8 (attributes must be available on the terminal).

Other restrictions:
For use with the Editor, a minimum baud rate of 4800 is recommended.

The terminal must use ASCII characters.

The last column in the last row on the screen cannot be used.

B.2 DISPLAY FILES

Maximum number of displays that can be contained in any given display file:
250

Maximum number of display files that can be used by an application program:
Limited only by disk space available on the run-time computer.

Maximum number of display files that can be open at any given time:
1

Maximum number of displays that can be in use at any given time:
1

B.3 FIELDS

Maximum number of fields permitted in a single display:
250

Maximum field size:
Cannot be longer than one row on the terminal screen.

Minimum field size:
one column.

Other restrictions:
Fields cannot overlap one another.

B.4 RUN-TIME LIBRARY

Memory required for typical displays, including program code for a 24 row by 80 column screen:
8K (additional memory required if there are a large number of fields in the displays).

Other restrictions:
The application program must pass the control codes for the run-time terminal to Display Manager, at run-time, using the INITDM function. The string passed by this function must not contain more than 254 characters.

The argument <string expression> passed by the PUTF function must not contain more than 132 characters.

Appendix C: Custom Terminal Setup

Any terminal used with Display Manager must have its terminal control codes correctly recorded in the TERMS.DM file. Otherwise, the terminal will not function as expected. The TERMS.DM file contains control codes for most terminals that you might use with Display Manager. However, there are situations where this is not the case, and the Custom Terminal Setup option in the DMSET program accommodates these situations.

This appendix explains how to complete a custom terminal setup. You might find this necessary in the following situations:

- TERMS.DM does not contain the codes for a design or run-time terminal that you want to use.

- TERMS.DM contains the codes for a terminal that you want to use but, due to the way certain switches or features are set on the terminal, the codes are incorrect.

- You want to remove control codes from TERMS.DM for a specific terminal.

- You want to examine the control codes for a particular terminal in TERMS.DM.

To perform a custom terminal setup, you need the manual for the terminal. The manual should provide all of the information you need to perform this procedure. If you do not have the manual, or cannot find the required information, you can contact your dealer for assistance.

Before performing a custom terminal setup, you might want to determine if terminal emulation is possible. Although the terminal that you want to set up might not be contained in the TERMS.DM file, it might use the same terminal control codes as one that is. In this situation, terminal emulation is possible. Simply select the terminal that you want to emulate from those in TERMS.DM and, if desired, select option C from the Custom Terminal Setup Options Menu (see the next figure) to correct the manufacturer and model name for the terminal to match yours. Thoroughly test the emulation, to ensure it functions properly (see Section 3.2.4, "Option T--Test Terminal Control Codes"). Appendix A lists the terminals in TERMS.DM. Use option E (EXAMINE terminal control codes) to review the actual control codes for any terminal in TERMS.DM.

To perform a custom terminal setup, start the DMSET program according to the instructions in Section 3. When the Main Menu appears, select option C. The Custom Terminal Setup Options Menu appears on your screen, as shown in the following figure.

```
-----------------------------------------------
      CUSTOM TERMINAL SETUP OPTIONS MENU
-----------------------------------------------
```

```
Option      Function
  T     Set up codes for THIS terminal
  F     Set up codes for a DIFFERENT terminal
  C     CHANGE existing terminal control codes
  D     DELETE terminal control codes
  E     EXAMINE terminal control codes
  X     EXIT (return to Main Menu)

Please enter your selection --> : :
```

Figure C-1. Custom Terminal Setup Options Kenu

The next table provides brief explanations of the functions available on this menu. The remainder of this section describes each procedure in detail. Section 3 and Appendix A also contain information pertinent to these procedures.

Table C-1. Custom terminal setup options

Format: Option
Explanation

Option T--Set up codes for this terminal
If TERMS.DM does not contain control codes for the terminal that you are now using, select this option to insert the codes.

Option F--Set up codes for a different terminal
If you want to set up codes for a terminal other than the one that you are now using, select this option.

Option C--Change existing terminal control codes
Select this option if you want to change the control codes for a user-supported terminal currently in TERMS.DM.

Option D--Delete terminal control codes
Select this option if you want to remove the terminal control codes for a user-supported terminal from TERMS.DM.

Option E--Examine terminal control codes
Use this option to look at any terminal control codes in TERMS.DM. You cannot make changes using this option.

Option X--Exit (return to Main Menu)
Self-explanatory.

C.1 OPTION T--SET UP CONTROL CODES FOR THIS TERMINAL

Use this option to set up terminal control codes for the terminal that you are now using. When you complete this procedure, a user-supported terminal entry is made in the TERMS.DM file for your terminal. You should then run the test procedure (option T on the DMSET Main Menu) to verify that the terminal functions as expected. The following figure "Set Up Control Codes for this

Terminal" illustrates this procedure.

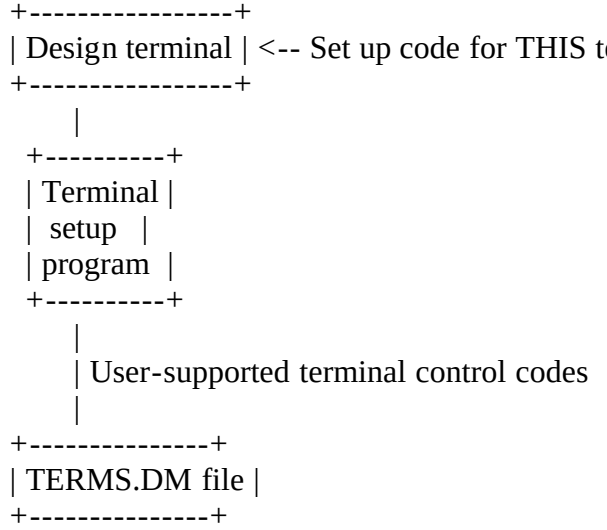


Figure C-2. Set Up Control Codes for this Terminal

When you select this option, the program asks you a series of questions about the codes that your terminal uses to perform such functions as clearing the screen, positioning the cursor, and others. You should be able to find the information needed to answer these questions in the manual for the terminal. If you do not have the manual, or are unable to determine the correct answer to the question(s), contact your dealer for assistance.

C.2 OPTION F--SET UP CONTROL CODES FOR A DIFFERENT TERMINAL

Use this option to set up terminal control codes for a terminal other than the one that you are now using. When you complete this procedure, a user-supported terminal entry is made in the TERMS.DM file for the terminal. You should then run the test procedure (option T on the DMSET Main Menu) to verify that the terminal functions as expected. Note that the terminal must be available to run the tests. The following figure "Set Up Control Codes for a Different Terminal" illustrates this procedure.

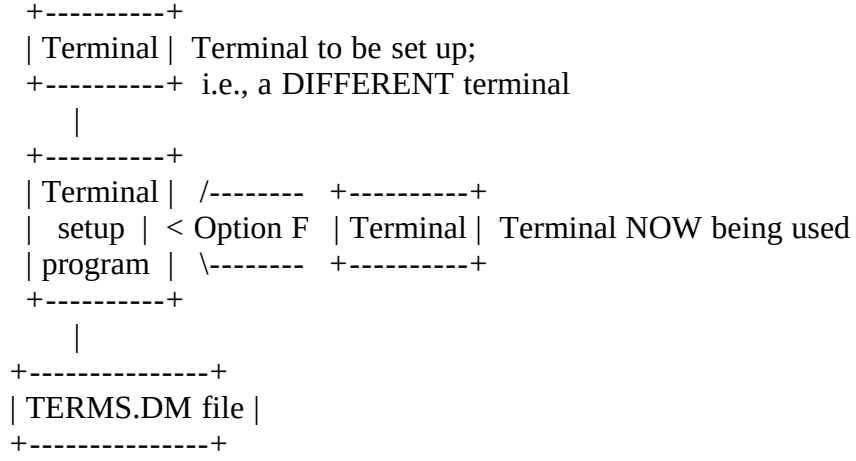


Figure C-3. Set Up Control Codes for a Different Terminal

When you select this option, you are asked to identify the terminal that you are now using to conduct this procedure. This is necessary, because displays created with Display Manager are used by this procedure to help you set up the required terminal control codes. You are shown the list of terminals in TERMS.DM, and asked to select the one that you are using by entering its associated 3-character code. If the code you enter is found in the list, the word "FOUND" appears, and you only need press RETURN to select it for use. If the 3-character code is not found, the words "NOT FOUND" appear, and you can use the DEL key ("<--") to erase your entry and enter a correct code. You can also use the scroll commands Ctrl-W and Ctrl-Z, or the ESC key, at this time.

If the control codes for the terminal that you are using to conduct this procedure are not contained in TERMS.DM, you have 2 options for continuing.

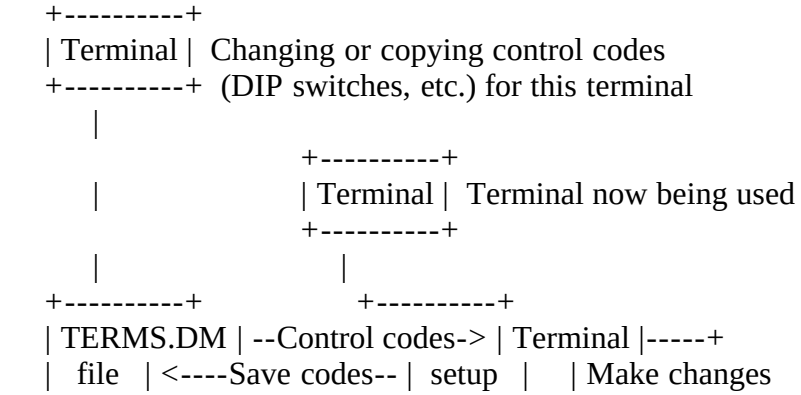
1. Press ESC to stop this procedure and return to the Custom Terminal Setup Options Menu. Select and complete option T to "Set up codes for this Terminal". Once you complete the procedure, this option is restarted for you automatically.

2. Select a terminal from the displayed list, that uses the same cursor positioning and clear screen control codes as the terminal that you are using. (These are the only 2 terminal functions used in this procedure.)

Once you have selected the terminal that you are now using, you are asked a series of questions about the terminal you want to set up. Section C.6, "Custom Terminal Setup Questions", at the end of this section, describes these questions.

C.3 OPTION C--CHANGE TERMINAL CONTROL CODES

Many terminals have special switches, usually called DIP switches, which can be set to alter the way one or more of the terminal features operate. For example, a switch can be set one way to activate a blinking line cursor, or another way to activate a steady box cursor. If the terminal that you are setting up has switches that have been set to alter the normal features of the terminal, you can use this option to create or change the control codes for the terminal to match the switch settings. The following figure "Change Terminal Control Codes" illustrates this procedure.



```
+----------+              | program  |<----+
                      +----------+
```

Figure C-4. Change Terminal Control Codes

As distributed, the TERMS.DM file contains the terminal control codes for all Display Manager-supported terminals. You cannot change these codes, but you can copy them and make changes to the copy. The copied codes with changes are set up in TERMS.DM as user-supported terminals. You can make as many changes as you like to the control codes for user-supported terminals. (See Appendix A for further information.)

When you select this option, the first thing that you are asked to do is identify the terminal whose control codes you want to copy or change. You are shown the list of terminals in the TERMS.DM file, and asked to enter the 3-character code of the one that you want to select. If you enter a code that is in the list, the word "FOUND" appears, and you must press RETURN to select that terminal. If you enter a code that is not in the list, the words "NOT FOUND" appear, and you can use the DEL key ("<--") to erase your entry and enter the correct code. You can also use the scroll commands Ctrl-W and Ctrl-Z, or the ESC key, at this time.

After you select the terminal that you want to change, you are shown the manufacturer and model name, and control codes, for that terminal.

In the second step, you are asked to select the terminal that you are NOW using by the same process explained in the preceding paragraph.

The third step is to make the actual changes to the terminal control codes. To accomplish this, the Custom Terminal Setup Questions Menu appears on your screen. Section C.6, "Custom Terminal Setup Questions, at this end of this section, describes these questions.

C.4 OPTION D--DELETE TERMINAL CONTROL CODES

Use this option to delete control codes for user-supported terminals from the TERMS.DM file. Note that you cannot delete control codes for Display Manager-supported terminals.

When you select this option, you are shown a list of user-supported terminals in the TERMS.DM file. User-supported terminals are indicated by (USER-SUPPORTED) displayed next to the terminal name and model. Each entry in the list has a 3-character code associated with it, such as 011 or 012. Scroll through the list until you find the terminal whose control codes you want to delete, then enter its corresponding 3-character code.

If you enter a code that is not in the list, the words "NOT FOUND" appear. Use the DEL key ("<--") to erase your entry, and enter a correct code. You can also enter the scroll commands Ctrl-W and Ctrl-Z, or the ESC key, at this point.

When you enter a correct code, the word "FOUND" appears. Press RETURN, and the

control codes for that terminal are displayed. You are then asked to confirm that this is the terminal that you want to delete. Finally, you are given the option to delete more codes, or return to the Main Menu.

When you delete codes from TERMS.DM, be sure to delete them from your table of user-supported terminals in Appendix A.4 (if you are maintaining this table).

C.5 OPTION E--EXAMINE TERMINAL CONTROL CODES

Use this option to look at how the control codes for one or more specific terminals are recorded in the TERMS.DM file. Note that you cannot write terminal control codes to a file, change, or delete them with this option. This option only allows you to look at the codes.

When you select this option, you are shown the complete list of terminals in the TERMS.DM file. User-supported terminals are indicated by (USER-SUPPORTED) displayed next to the terminal name and model. Each entry in the list has a 3-character code associated with it, such as A41 or Z11. Scroll through the list until you find the terminal whose control codes you want to examine, and then enter its corresponding 3-character code.

If you enter a code that is not in the list, the words "NOT FOUND" appear. Use the DEL key ("<--") to erase your entry, and enter a correct code. You can also enter the scroll commands Ctrl-W and Ctrl-Z, or the ESC key, at this point.

When you enter a correct code, the word "FOUND" appears. Press RETURN, and the control codes for that terminal are displayed on your screen. You are then given the option to examine more codes, or return to the Main Menu.

C.6 CUSTOM TERMINAL SETUP QUESTIONS

This section explains the questions that you are asked when you select option F or C from the Custom Terminal Setup Options Menu (shown earlier in this Section, in Figure C-1). There are 10 different categories of questions, with each relating to a specific terminal feature. These question categories appear on a menu as shown in the next figure.

```
-------------------------------------------------------
      CUSTOM TERMINAL SETUP QUESTIONS MENU
-------------------------------------------------------
Code       Category
 1 . . .  Screen Size
 2 . . .  Clear Screen
 3 . . .  Cursor Positioning
 4 . . .  Startup Codes
 5 . . .  Standard Video Attributes
 6 . . .  User-defined Attributes
 7 . . .  Multiple Attributes
 8 . . .  Cursor Arrow Keys
```

```
 9 . . . Function Keys
 A . . . Cursor ON/OFF
 H . . . HELP
 S . . . Save Terminal Control Codes
ESC . . . Return to Main Menu
Please enter your selection --> : :
```

Figure C-5. Custom Terminal Setup Questions Menu

You must answer questions in these categories:

1 ... Screen Size
2 ... Clear Screen
3 ... Cursor Positioning
4 ... Startup Codes

Answers to the remaining categories are optional.

Most questions provide a default answer, which is enclosed in the signs < >. If you are unsure of an answer, try using the default. However, be sure to verify that the default works by running the test procedure from the DMSET Main Menu.

Four types of questions can occur, each requiring a specific type of answer:

1) Yes/No Questions require a response of Y or N.

2) Plant/Paint Questions require a response of L (meaning a pLant attribute is used) or A (meaning a pAint attribute is used). Section 4 describes Plant and Paint attributes.

3) Numeric Questions require an integer value in a decimal format for the answer.

4) Character Sequence Questions are answered by entering a sequence of ASCII, decimal, or hexadecimal characters. You can enter up to 8 characters. These characters sequences activate certain features on some terminals.

Once you answer a category of questions from the Custom Terminal Setup Questions Menu, the program places parentheses around that category in the menu, to indicate that changes have been made. However, remember that your changes are not saved in the TERMS.DM file until you select and complete the S option from the menu.

The following sections explain the questions asked in each category, and provides space for you to record your answers.

When running the program, you can select categories from the menu in any order. As you read through the terminal manual, select a category and answer the questions as you find the required information. Note that all questions listed in a category might not necessarily appear on your screen; many questions depend on the answer given to a previous question.

C.6.1 Screen Size Questions

The answers to these questions indicate how many rows and columns are available on the terminal screen.

How many ROWS are on the screen? (Top to bottom) <24> ____

How many COLUMNS are on the screen? (Left to right) <80> ____

C.6.2 Clear Screen Questions

You must tell Display Manager how to clear the screen to all blank characters. Some terminal manuals refer to this as "clear page" or "home cursor".

What characters must be sent to the screen to clear it? ____

How many milliseconds delay is needed after clearing the screen? <50> ____

C.6.3 Cursor Positioning Questions

Terminals can directly position the cursor in a specific row and column of the screen when sent the correct information. The following questions ask you for the information Display Manager must send to the terminal to position the cursor.

To position the cursor, is the ROW number sent first? (N/Y) <Y> ____

When the cursor moves right, does the COLUMN number increase? (N/Y) <Y> ____

When the cursor moves down, does the ROW number increase? (N/Y) <Y> ____

What COLUMN number positions the cursor at the left edge? ____

What ROW number positions the cursor at the top edge? ____

What characters must be sent PRECEDING the row and column number? ____

What characters (if any) must be sent

BETWEEN the row and column number? ____

What characters (if any) must be sent
FOLLOWING the row and column number? ____

Are the row and column numbers sent as
SINGLE, INDIVIDUAL bytes? (N/Y) <Y> ____

(If previous answer was No...)
Are row and column numbers sent as
a STRING of characters? (N/Y) <Y> ____

(If previous answer was No, the terminal
cannot be set up. If answer was Yes...)
How many characters in the STRING? <3> ____

How many MILLISECONDS delay is
needed following a cursor move? <30> ____

C.6.4 Start-up Codes Questions

A string of start-up codes must be sent to some terminals, to place them in their normal operating mode. This ensures that they can be used with Display Manager. You must provide the start-up codes for the terminal that you are setting up if it does not start in full-intensity, normal video, with only remote key operation (that is to say: the screen is controlled by signals sent from the computer, rather than from the keyboard). The following questions ask for the start-up codes.

What CHARACTER STRING (if any)
places the terminal in normal mode? ____

What CHARACTER STRING (if any)
places the terminal in default mode? ____

C.6.5 Standard Video Attributes Questions

Video attributes are visual effects such as half-intensity or reverse video that the terminal can perform on information shown on the screen. A string of up to 8 characters can be sent to activate or deactivate a video attribute. Section 4.4 describes video attributes.

	+---------- Feature ----------+	
	Half Intensity	Reverse Video	Flashing	Underlining
Is this feature AVAILABLE? (Y/N) <N>	_____	_____	_____	_____

Does it require a
PAINT or PLANT

attribute? (A/L) <L> _____ _____ _____ _____

What CODE ACTIVATES
the feature? _____ _____ _____ _____

What CODE DEACTIVATES
the feature? _____ _____ _____ _____

C.6.6 User-defined Attributes Questions

Any feature available on the terminal can be activated as a user-defined attribute, as long as it functions like a standard video attribute. That is to say: it only affects text characters, can be turned OFF, and does not move the cursor. Up to 3 different user-defined attributes can be used with Display Manager.

	User Attr#1	User Attr#2	User Attr#3
Is this attribute USED? (Y/N)	<N>	_____	_____	_____
Does it require a PAINT or PLANT attribute? (A/L)	<L>	_____	_____	_____
What CODE ACTIVATES the feature?		_____	_____	_____
What CODE DEACTIVATES the feature?		_____	_____	_____

C.6.7 Multiple Attributes Questions

If the terminal has the capability, Display Manager can activate 2 separate video and/or user-defined attributes simultaneously. There are 2 methods for doing this: a special code might exist for the combination, or the 2 attributes might be activated as separate features. In the latter case, at least one attribute must be a Paint-type, because only one column can be reserved on either side of a field. To use the following table, mark each combination of attributes that can be activated simultaneously, and answer the corresponding questions.

Combinations							Questions			
Half Ints	Rever Video	Flash line	Under #1	User-defined #2	#3	Special Code? (A/L)	Paint /Plant Code	Acti- vate Code	Deacti- vate Code
____	_____	_____	____	___	___	___	<N>	____	_____	____	____
____	_____	_____	____	___	___	___	<N>	____	_____	____	____
____	_____	_____	____	___	___	___	<N>	____	_____	____	____

____ _____ _____ ____ ___ ___ ___|<N> ____ _____ ____ ____
____ _____ _____ ____ ___ ___ ___|<N> ____ _____ ____ ____
____ _____ _____ ____ ___ ___ ___|<N> ____ _____ ____ ____

C.6.8 Cursor Arrow Keys Questions

Some terminals have special keys with arrows marked on them for cursor movement. They can be used with Display Manager if they produce either the ANSI standard arrow characters (Ctrl-H, Ctrl-J, Ctrl-K, or Ctrl-L) or the Display Manager cursor movement characters (Ctrl-S, Ctrl-E, Ctrl-D, or Ctrl-X).

Does the terminal have
cursor arrow keys? (Y/N) <N> ____

(If the previous answer was Yes...)
Are standard cursor arrow keys
(^H,^J,^K,^L) used? (Y/N) <N> ____

Are Display Manager cursor arrow
keys (^S,^E,^D,^X) used? (Y/N) ____

C.6.9 Function Keys Questions

There are 2 types of function keys. Display Manager cannot be used with the type that produces local effects (such as Scroll or Delete). Display Manager can be used with the type that sends a sequence of characters when pressed. In the following questions, "value" refers to the ASCII character that is transmitted when the function key is pressed.

How MANY function keys are there?
(with same leading/trailing characters) <0> ____

(If previous answer is >0...)
What LEADING characters must
be sent to the screen? ____

What TRAILING characters must
be sent to the screen? ____

What is the VALUE of the FIRST key
in the function key set? ____

What is the VALUE of the LAST key
in the function key set? ____

C.6.10 Cursor ON/OFF Questions

The cursor can be made invisible on some terminal screens. Note that some terminals have several possible shapes for the cursor. In such cases, the code to turn the cursor ON should be the most commonly-used one.

```
Can the cursor be turned
ON and OFF? (Y/N)              <N> ____

(If previous answer was Yes...)
What characters must be sent
to turn the cursor OFF?         ____

What characters must be sent
to turn the cursor ON?          ____
```

C.7 COMPLETING CUSTOM TERMINAL SETUP

After you have answered all of the questions for the terminal that you are setting up, select the S option on the Custom Terminal Setup Questions Menu to save the information and record the control codes in TERMS.DM. When you select this option, a display similar to the following figure appears on your screen:

```
User-supported terminal number: 01
Control code:
/1ABADBKBLCKDCCACACAACBLDNAAAAAABIFAAAAAU3
Enter Manufacturers name:
Enter Model Name:
```

Figure C-6. User-supported Terminal Setup Screen

This display shows you the number assigned to this user-supported terminal (01) and its control codes (/1ABAD...). The first prompt asks you to enter the manufacturer's name for the terminal, and the second asks for the model name. Whatever information you enter here appears on the screen when the Editor program is started. However, you can suppress this information by placing square brackets ("[" and "]") around your response. After you enter the information, it is stored in the TERMS.DM file, along with the control codes. (Note that space is provided in Appendix A to record this information for future reference. See Section A.4.)

At this point, you are returned to the DMSET Main Menu.

Glossary

application program:
A series of coded instructions telling a computer how to process information it receives. The written code is referred to as the source code. The source code is submitted to a compiler which translates the code into instructions that can be understood by the computer. The translated code is referred to as the object module.

border mode:
While in border mode, you can use certain commands to instruct the Editor to draw borders and other figures on the design terminal screen. See Section 6.

boundary mode:
While in boundary mode, you can use certain commands to instruct the Editor to show where the boundaries are for input and output fields in a display. The boundaries indicate the position, length, and type of each field. See Section 6.

color attributes:
Some computer's terminals have the ability to show displays in color. You can define input and output fields in a display to use these features, when available, by assigning the appropriate color attributes in the field's status window, or as global values for the display. You can designate separate colors for backgrounds and foregrounds. See Section 4.

CRT:
Cathode Ray Tube. The CRT is the part of a computer terminal that looks like a television screen.

Ctrl key:
The Control key on a terminal keyboard. You enter most Editor commands by holding down the Ctrl key while simultaneously pressing another key. Sometimes, the caret symbol (^) is used to abbreviate Ctrl. For example, both Ctrl-Z and ^Z means: "type the letter Z while holding down the Ctrl key".

current display:
A term used in the Editor environment to reference the display that is currently being created or edited.

current display file:
A term used in the Editor environment to reference the display file that is currently open.

current field:
Field in a display that contains the cursor. You can use the POSF or NXTF function in an applications program to place the cursor in a field.

cursor control keys:
Keys used to control movement of the cursor on the screen of the design or run-time terminal. These are sometimes referred to as cursor arrow or cursor movement keys. There are 2 sets of cursor control keys: the standard ANSI set (Ctrl-H, Ctrl-J, Ctrl-K, and Ctrl-L) and the standard editing keys (Ctrl-S, Ctrl-E, Ctrl-D, and Ctrl-X). See Section 6 and the INITDM function description in Section 7.

DEL key ("<--"):
ROCHE> This key "deletes the character to the left of the cursor". Unfortunately, DEL has at least 3 different meanings. 1) The ASCII 7F hex character, that was used on ASR-33 Teletype to DELete the preceding character by setting it to all ones (hence its location at 7F hex, since ASCII is a 7-bit code). 2) The ASR-33 Teletype key RUB OUT (or RUBOUT) that was later sometimes called "DEL", sometimes "BACKSPACE". 3) The IBM PC had a DEL key on

its keyboard until someone noticed that another key (used in the famous "Ctrl+Alt+Del") sequence also had the same name... As a consequence, it was "renamed" "<--", loosing its name in the process... Normally, historically, this key was called "RUB OUT" on the ASR-33 Teletype, but 99.99% of persons using a "IBM Clown" computer do not understand the difference between erasing (ASCII BACKSPACE 08 hex) and deleting (ASCII DEL 7F hex). So, I chose to use the name that became common with the IBM Clown, with its symbol ("<--"). I hope to have been clear.

design terminal:
Terminal used with the Editor to create displays subsequently used in application programs.

display:
Image shown on the screen of a design or run-time terminal. You create displays with the Editor, and subsequently use them in your applications program to interact with the end-user.

display field:
Part of the display into which information is either entered by or shown to the end-user. These can be literal, input, or output fields.

display file:
A disk file containing a group of related displays created with the Editor. Each display file can contain up to 250 displays. An application program can use as many different display files as disk space on the run-time computer permits. As an informal standard, display filenames have a filetype of DIS, such as DISPLAYS.DIS.

display reference number:
Number ranging from 1 to 250 that you assign to each display when creating it with the Editor. When you want to retrieve a display for use with the Editor or with the application program, you reference it using this number. While using the Editor, you see a list of the displays in a display file with their assigned display reference numbers.

editor:
Program you run on your computer to design, create, change, and delete displays. The Editor must be created for use with your specific design terminal before it can be used (see Section 3 for instructions). Sections 4 through 6 contain complete instructions for using the Editor.

end-user:
A person who runs your application programs on a run-time computer. In Display Manager, a clear distinction exists among the person designing the displays, the person creating the application programs, and the end-user.

field reference number:
Every field in a display is assigned a field reference number when you create it with the Editor. Numbers can range from 1 to 250. The Editor provides a command for renumbering one or more fields in a display. The POSF function positions the cursor in a specific field when passed the field reference number.

function:
A function is a routine in the Display Manager Run-time Library that is used at run-time to manage displays and display fields. These functions are called out of the Run-time Library by statements embedded in the source code of the application program. The function call must conform to a specific syntax. Section 7 contains descriptions of all Display Manager functions and instructions for using them.

function key:
Keys than can be programmed to perform specific, predetermined functions. You can instruct Display Manager to react in certain ways when one of the function keys is used. Not all terminals are equipped with function keys.

initial value:
Input and output fields in a display can be assigned an initial value. This value appears in the field when it is initially shown in the display (unless the field has been assigned the "invisible" attribute). If the UPDF function is used with a field having an initial value, the value is returned to the application program if the end-user does not enter any other information into the field. On the other hand, if the GETF function is used, a null string is returned to the application program if no information is entered.

input field:
A display field in which an end-user types information. For example, an input field might be used to enter a customer name, a selection from a menu, and so forth. Information can also be placed in an input field to be shown to the end-user. Section 6.2 (Ctrl-UI) explains how to create input fields. Section 7 describes the functions you can use to manage the fields in a display.

literal field:
Any constant information appearing in a display. It usually acts as a label for an input or output field, provides instructions for the end-user, or serves as a prompt.

output field:
A display field used to show information to the end-user. You can specify a format, when the field is created, indicating how the information should appear. Section 6.2 (Ctrl-UO) explains how to create output fields. Section 7 describes the functions you can use to manage the fields in a display.

run-time:
When an application program is run on the end-user's computer. Section 8 describes the run-time environment.

run-time errors:
Errors that can occur at run-time as a result of using one of the functions in the Run-time Library. The end-user will never encounter these errors if you have adequately tested the application program, and included the logic necessary to intercept such errors. See Section 8.

Run-time Library:
A library of routines provided on your Display Manager distribution disks. Routines (functions) are called from the library when your application program needs to manage displays or fields in a display. You must link your

application program object module to the Run-time Library to use the routines contained therein. Your "Display Manager Programmer's Guide" contains linking instructions.

run-time terminal:
Terminal used at run-time to show displays created with the Editor.

status window:
Every input and output field in a display has a status window associated with it. You can use Editor commands to show the status window on your screen while designing the display, and move the cursor inside the status window to define the characteristics of a display field. Section 4.6 describes the status window.

template characters:
You can use templates in display fields to simplify data entry for the end-user. They indicate exactly how information must be entered in a field. Template characters are never returned to your application programs. See Section 4.

terminal control codes:
A string of codes sent to a terminal to make it do what it is supposed to do. The codes also tell an application program how to send information to the terminal to do such things as clear the screen, position the cursor, and more. See Appendix A.

video attributes:
Most computer terminals have one or more optional features such as reverse video, half intensity, underlining, and others. You can define input and output fields in a display to use these features, when available, by assigning the appropriate video attributes in the field's status window. For example, you can set the video attribute of a field so that, when it appears in the display, it appears in inverse video. See Section 4.

Index

(To be done by WS4...)

EOF