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 runtime. 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+, 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 runtime.

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-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
```

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 nn 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 runtime, 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
Position the cursor to a given field.

NXTF
Positions the cursor to the next field.

SETF
Modifies the visual attributes of a field during runtime.

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 aguments. 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 runtime 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 runtime package that allows non-dBase II owners to use the package. Unfortunately, the dBase II runtime 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